Evolving Scheduling Heuristics with Genetic Programming for Optimization of Quality of Service in Weakly Hard Real-Time Systems

Karla Salamun[a,*], Ivan Pavić[a], Hrvoje Džapo[a], Marko Đurasević[a]

[a] *University of Zagreb Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia*

Abstract

The weakly hard real-time system model is used for describing the real-time systems that allow occasional violations of real-time timing constraints. These systems include real-time control systems, multimedia systems, and communication systems. In some approaches that deal with mitigating the system overload in real-time systems with periodic tasks, namely job-skipping algorithms, the constraints defined by the weakly hard real-time model are used as a mechanism for defining the pattern of task instances (jobs) that may be skipped in order to reduce the system load. The performance of these algorithms is usually evaluated with respect to the quality of service metric, which depends on the number of skipped jobs. In this work, we investigate the possibility of using genetic programming in the automated synthesis of scheduling heuristics for optimizing skipping patterns in order to increase the average quality of service in comparison with the conventional job-skipping algorithms. Using genetic programming to automatically synthesize heuristics allows for an easy and quick design of novel heuristics for various problem types and optimization criteria. We present two different approaches for implementing the proposed method. The first approach is to encapsulate the evolved heuristics into job-skipping algorithms known from the literature, namely Red Tasks as Late as Possible (RLP) and Blue When Possible (BWP). The idea of the second approach is to employ the evolved heuristics as

*Corresponding author

Email addresses: karla.salamun@fer.hr (Karla Salamun), ivan.pavic2@fer.hr (Ivan Pavić), hrvoje.dzapo@fer.hr (Hrvoje Džapo), marko.durasevic@fer.hr (Marko Đurasević)

standalone job-skipping algorithms. The results show an improvement of up to **15%** in comparison with the state-of-the-art algorithms. The novel methods described in this work present a significant upgrade of the standard job-skipping algorithms as they provide a notable improvement in terms of quality of service while ensuring the fulfillment of weakly hard constraints. Moreover, the presented methods are computationally efficient and are therefore suitable for implementation on real-time operating systems, which is not the case with similar methods based on optimization techniques.

Keywords: weakly hard real-time systems, genetic programming, scheduling, overload

1. Introduction

Handling overloaded conditions in various real-time computer systems is an active research area. Overload in real-time systems is defined as a condition in which the computational demand requested by the application exceeds the processor capacity [1]. Real-time systems are typically modelled as sets of periodic or sporadic tasks which release an infinite sequence of task instances that are referred to as jobs. The released jobs are placed in a queue, where they wait to be scheduled for execution. In an overloaded real-time system, the corresponding tasks are unable to fulfill their real-time timing requirements—the computation required by each job cannot be completed by its respective deadline. In certain real-time applications, such as control systems or multimedia streaming systems, providing response after the deadline is referred to as *skipping* or *missing* the deadline, since late response brings no significance to the system. The severity of consequences caused by missing job deadlines depends on the target application of the real-time system. In this paper, we consider real-time systems that allow a certain degree of missing deadlines as long as they occur in a bounded and predictable manner, i.e., *weakly hard* real-time systems [2]. Weakly hard real-time systems can tolerate deadline misses either because their robust design ensures that the stability of the system will not be jeopardized by missing some deadlines, or deadline misses are not noticeable by the user if a certain degree of timely completed tasks is ensured. However, certain timing constraints must be fulfilled in order to ensure the stability and satisfactory performance of the system. The timing constraints that specify acceptable ratios and distributions of skipped deadlines in weakly hard real-time systems are referred to as *weakly hard constraints* [2].

In this work, we deal with systems that have a constraint on the minimal number of subsequent jobs which must be completed for every job that is skipped. In other words, it is only allowed to skip a single task instance, i.e., job, in a given number of consecutive jobs. The job that may be skipped without violating this constraint is referred to as *skippable job*. This task model is known in the literature as *skip-over* model [3]. According to the skip-over model, tasks are periodic and preemptable, and there are no precedence constraints among the tasks. The considered real-time system model is typically applied in real-time control systems, monitoring systems, and communication systems. These applications are mostly implemented on embedded computer platforms with limited computational and memory capacity. These constraints must be taken into account upon selecting the algorithm for scheduling tasks. It is often required for the scheduling algorithm to run on-line, and therefore complex algorithms that introduce computational overhead are not suitable for these applications.

The problem of scheduling tasks under the skip-over model comes down to constructing an algorithm that guarantees the satisfaction of the constraints provided by the skip-over task model. In overloaded conditions, it is certain that some of the jobs will be skipped, and therefore the performance of the scheduling algorithms is usually evaluated with regard to the obtained quality of service, i.e., the ratio of completed jobs in the total number of released jobs.

There are several existing approaches that address the problem of scheduling in real-time systems under skip-over model [3, 4, 5]. In this work, we interpret these approaches as *meta-algorithms*. A meta-algorithm defines how the jobs are scheduled according to the system state and the constraints defined by the given scheduling environment [6, 7]. The meta-algorithms encapsulate a *priority function*, i.e., a function that assigns priority values to jobs, considering the properties of the respective job and the overall system state. In the considered context, a meta-algorithm provides a mechanism that guarantees that the generated schedule will fulfill the constraints defined by the skip-over model. More precisely, a meta-algorithm ensures the timely completion of jobs that are crucial for the fulfillment of constraints given by the skip-over model. These jobs are referred to as *mandatory* jobs and the jobs that may be skipped are referred to as *skippable* jobs. The priority function defines an expression for calculating the priorities of the jobs, regardless of whether the job is mandatory or skippable.

The scheduling problem that arises from the skip-over model can be viewed as an optimization problem—the goal is to design a scheduling meta-algorithm or standalone scheduling heuristic that maximizes the obtained quality of service while guaranteeing the fulfillment of skip-over constraints. In the literature, methods for solving optimization problems that proceed from scheduling problems often rely on machine learning techniques [8]. Nowadays, machine learning is present in numerous fields of scientific research, as it is widely used for solving prediction [9] and classification [10] problems. Due to the high complexity and stochastic nature of most scheduling problems, the most commonly-used approaches for solving these problems are heuristic methods for optimization, specifically, hyper-heuristic methods [11]. The main idea of these methods is to automate the design of scheduling heuristics that solve a specific scheduling problem. The most popular area of research focus for hyper-heuristic approaches is production scheduling [12], and genetic programming is the dominating technique [13].

In this paper, we investigate the possibility of evolving custom priority functions by genetic programming with the aim of increasing the quality of service. The evolved priority functions are computationally efficient, and therefore the methods presented in this work are suitable for implementation on real-time embedded systems. To the best of our knowledge, our approach is the first to apply the genetic programming technique and hyper-heuristic approach to scheduling problems derived from the weakly hard real-time system model. Moreover, it is one of the few approaches in general to apply the hyper-heuristic technique to scheduling problems in real-time systems, as previous research is mainly focused on production scheduling. The key difference between production scheduling problems and real-time scheduling problems is that real-time systems do not allow job execution after its respective deadline—tardy jobs are discarded. On the other hand, job tardiness is allowed in production scheduling problems. Moreover, in weakly hard real-time systems, there are additional constraints that specify jobs that must not be discarded, making the scheduling problem even more specific.

We present and discuss two different variants of the approach. Firstly, we consider encapsulating the evolved priority functions into existing meta-algorithms for scheduling tasks under the skip-over model. In the paper, we refer to this approach as *GP-QoS-MA*. The goal of this approach is to evolve a generalized algorithm that can be applied to any problem instance. The GP-QoS-MA algorithm relies on the skip-over meta-algorithm in order to ensure the fulfillment of weakly hard real-time constraints while aiming to maximize the quality of service through a best-effort heuristic for scheduling skippable jobs. The other approach is to evolve the priority function as a scheduling heuristic that is not incorporated into any meta-algorithm. We denote this approach as *GP-QoS-S*. However, by adequately modifying the optimization process, the evolved heuristic can behave as the considered job-skipping meta-algorithms in terms of guaranteeing the satisfaction of constraints provided by skip-over task model. The GP-QoS-S approach is applicable in a scenario where we deal with a predefined scheduling problem and we need to design a scheduling heuristic that is optimized for the corresponding task set and a given metric. Although the scheduling problem for a single task set can be solved by search-based techniques, e.g. genetic algorithms [14, 15], there are strong arguments in favor of employing genetic programming in the application considered in this work. Generally, the output of the search-based techniques is a schedule that determines the execution of tasks in a given time interval, which needs to be statically stored in the memory on the target real-time system. The time interval of interest typically corresponds to the *hyperperiod* of the task set: a time interval after which the schedule repeats itself. The hyperperiod can be extremely large and consequently the schedule can consume an immense amount of memory, which is not feasible in the context of embedded systems since they typically have limited memory resources [16]. Moreover, in overloaded conditions that are addressed in this work, the schedule does not repeat itself after the hyperperiod and therefore it is not possible to find a time interval such that the entire schedule can be stored in the memory [2]. Therefore, in dynamic and uncertain conditions it is far more appropriate to employ the genetic programming technique as it searches the space of algorithms that provide solution to the scheduling problem instead of searching the space of solutions, i.e., schedules.

The main contributions of this work can be summarized as follows:

- We introduce a method for designing priority functions using genetic programming, which can be integrated into conventional scheduling algorithms for skip-over system models, with the aim of maximizing the quality of service (GP-QoS-MA).
- We present and discuss a method for designing standalone scheduling heuristics using genetic programming for specific problem instances in the context of the skip-over system model, with the aim of maximizing the quality of service. We demonstrate how the obtained heuristics can learn the behavior of conventional job-skipping algorithms in the sense

of fulfilling the weakly hard constraints (GP-QoS-S).

- We provide systematic evaluation of the proposed methods and comparison to existing conventional approaches.
- We suggest the possibility for the implementation of the presented approaches in real-time embedded systems and evaluate the feasibility of the suggested methods with regard to the computational complexity of the generated heuristics.

It is important to emphasize that the presented approach can be easily customized for different scheduling environments and for optimizing different performance metrics.

The rest of the paper is organized as follows. Section 2 outlines the related work and gives practical examples that illustrate the motivation for this research. Section 3 describes the considered system model and the corresponding scheduling problem. The genetic programming method for evolving scheduling heuristics is described in Section 4. The setup for experimental evaluation and the obtained results are described in Section 5. Suggestions on how the presented methods can be applied in real-time embedded systems are given in Section 6. Section 7 summarizes the most important results and contributions and describes potential directions for future research.

2. Background

2.1. Related work

Our research is related to several approaches for designing scheduling algorithms for overloaded real-time systems and using genetic programming for evolving scheduling heuristics. Since the approach presented in this work is related specifically to weakly hard real-time systems, we will give an overview of the approaches from the literature that share the common task model.

The earliest research associated with real-time systems that allow occasional deadline violations is the work of Hamadaoui and Ramanathan, in the context of scheduling messages in communication systems. In [17], they introduced the concept of *(m, k)-firm* deadlines as an approach for modeling tasks that must meet m deadlines in every k consecutive jobs.

In [2], the authors present the weakly hard model of a real-time system as a framework for specifying real-time systems that tolerate occasional deadline misses. According to this model, the restrictions regarding the ratio and distribution of skipped jobs in a given number of jobs are classified as four constraints. The approach described in this work is related to the constraint $\left\langle {m \atop k} \right\rangle$ (meets row m in k deadlines) which specifies that in any window of k consecutive invocations of the task, at least m consecutive invocations must meet their deadlines.

The techniques for mitigating the effects of overload in real-time systems are reviewed in [18]. There are two types of overload in real-time systems: transient overload that occurs due to task overruns and permanent overload that ocurrs in periodic task systems, when the total utilization of the task set is greater than one. Since this work targets periodic task systems, we will review the approaches that deal with permanent overload only. The existing approaches for reducing permanent overload are based on one of three methods: *job skipping*, *period adaptation*, and *service adaptation*. The period adaptation technique is based on regulating the effective utilization factor by enlarging the task periods [19]. Service adaptation aims to reduce system load by reducing the computational time required by the tasks [20]. Since our work focuses on job skipping technique, we will describe this method in more detail in the rest of the paper.

The main idea of the job skipping technique is to reduce the effective utilization of the system by skipping a certain number of jobs and therefore enabling to schedule task sets which would otherwise be infeasible. Job-skipping algorithms provide a mechanism that guarantees that a minimum number of jobs per task will execute within their deadlines. This approach is suitable for applications where occasional deadline violations do not harm the performance or the stability of the system as long as the violations occur according to a predefined pattern, e.g., control applications or multimedia systems. The job skipping technique is based on the skip-over task model, which was introduced by Koren and Shasha in [3]. The authors introduced the *skip factor* parameter S_i as a specific case of $\left\langle {m \atop k} \right\rangle$ weakly hard constraint where $m = k - 1$. More precisely, the skip factor notation corresponds to weakly hard constraint $\left\langle {S_i - 1 \atop S_i} \right\rangle$. Skip factor ensures that every job that missed its deadline will be followed by at least $S_i - 1$ timely completed jobs. The authors proposed a model for classifying jobs depending on whether the deadline of the current job can be skipped by declaring the job state as *red* or *blue*. A red job must complete its execution before the deadline, while a blue job can miss its deadline. Koren and Shasha introduced two job-skipping algorithms, namely *Red Tasks Only* (RTO) and *Blue When Possible* (BWP). The implementation of these algorithms will be described in detail in Subsection 3.3. The skip-over model was further investigated by Marchand and Silly-Chetto in [4] where they introduced a new and more efficient algorithm *Red Tasks as Late as Possible* (RLP). This algorithm relies on scheduling skippable jobs in spare processing time given by non-skippable jobs. Since this approach requires on-line computation of time intervals that correspond to idle time given by non-skippable jobs, it is not suitable for implementation on systems with limited resources due to high computational complexity. The state-of-the-art techniques based on job skipping can be improved by introducing dispatching rules designed specifically for skip-over task model that maximize the number of completed jobs.

The majority of approaches that deal with designing dispatching rules for scheduling problems are based on hyper-heuristic methods. The most commonly-used method is genetic programming, as it has shown many advantages over other hyper-heuristic approaches such as decision tree [21], logistic regression [22], support vector machine [23], and artificial neural networks [24]. Unlike mentioned approaches, genetic programming provides flexibility for representation of the heuristics, it can simultaneously explore both the structure and the parameters of a heuristic, and it supports multi-objective optimization. Moreover, heuristics obtained by genetic programming are easily interpretable and computationally efficient [13], [25]. The research presented in [26] describes how existing scheduling heuristics can be improved in terms of schedule makespan by employing a reinforcement learning agent.

In the following text, we provide a brief overview of the recent work that employs genetic programming for evolving priority rules for different scheduling environments to show that this is an active research area. The approaches that employ genetic programming in production scheduling are covered in an extensive survey in [13]. The research presented in [7] describes how genetic programming may be applied to arbitrary scheduling evironments by encapsulating the evolved heuristic into a meta-algorithm that corresponds to the given environment. The authors demonstrate that the heuristics synthesized by genetic programming dominate multiple common approaches, e.g., *Earliest Due Date* (EDD), *Weighted Shortest Processing Time* (WSPT). The overview of the approaches that apply genetic programming in job shop scheduling is given in [27]. The application of genetic programming in parallel machines environment is described in [28, 29]. In recent years, the trends in designing the dispatching rules by genetic programming are focused on constructing ensembles of dispatching rules, with the aim of improving performance [30, 31].

The single-machine environment, which is of interest in this work, was investigated in [32], [33], [34]. The authors proposed methods for evolving scheduling policies in the form of dispatching rules with the objective of minimizing the total tardiness. In [35], similar approach was used, with an additional objective that minimizes the average wait time for the jobs. In a recent research related to single-machine scheduling, genetic programming is used for developing priority rules in the context of scheduling a fleet of electric vehicles arriving to a charging station [36]. The problems addressed in the described research papers can be categorized as production scheduling problems. The corresponding problem formulation allows jobs to be tardy, and hence the goal is to minimize tardiness. Our work focuses on a different problem formulation—tardy jobs are discarded, and the objective is to maximize the quality of service. In [37], the single machine environment is extended to the real-time mixed-criticality systems, and it is presented how genetic programming may be used for generating dynamic priority assignment functions for scheduling low-criticality tasks. The considered system model is similar to the one described in this work; however, in [37], the restrictions on jobs are formed in terms of criticality, whereas in this work, the restrictions are defined through weakly-hard real-time constraints, i.e., jobs are either mandatory or skippable.

2.2. Motivation

In this section, we provide practical examples that illustrate the motivation for analyzing weakly hard real-time systems. Moreover, we demonstrate how introducing weakly hard constraints into the system model and using adequate scheduling algorithms can improve performance in overloaded conditions.

2.2.1. Control systems

A typical example of a weakly hard real-time system in the industrial context is a computer-driven control system [38]. Computer-driven control systems can be modelled as sets of periodic real-time tasks, where each task carries out a single functionality of a control loop, e.g., samples the data from sensors, executes a control algorithm or sends commands to the actuators. The frequency of task invocations is determined by the sampling frequency of the input signal. Moreover, it is suggested that the sampling frequency should be between 5 to 10 times larger than the frequency demanded by the sampling theorem, since the control system can benefit from oversampling

Table 1: Parameters of the subsystems in an automotive control system.

Functionality	Task	Processor	Utilization factor (u_i)	Criticality
Engine control	τ_1	$\mathcal{P}_1$	0.3	high
Antilock braking control	τ_2	$\mathcal{P}_1$	0.25	high
Traction control	τ_3	$\mathcal{P}_2$	0.25	high
Cruise control	τ_4	$\mathcal{P}_2$	0.25	medium

[39, 40]. Each task invocation (job) has a deadline by which it must complete its execution. Typically, the deadline of the current job corresponds to the activation time of the following job. In some conditions, such as processor failure or arrival of aperiodic requests, the system can become overloaded and that causes the deadline miss for some jobs. In this example, we will analyze the behavior of a control system under overload in two different cases:

- in the first case, the scheduling algorithm discards jobs according to a precisely defined pattern,
- in the second case, the jobs are discarded randomly with a predefined probability.

In practice, many control algorithms implement advanced techniques such as reinforcement learning [41], [42], evolutionary computation [43], [44], and fuzzy logic [45], [46]. The weakly hard real-time model can be applied to any type of control system, regardless of the implemented control algorithm. However, as this paper addresses scheduling problems rather than control problems, the control system in this example will be based on a PID regulator for simplicity.

Example 2.1. *Consider an automobile control system that consists of four subsystems: engine control, antilock braking control, traction control, and cruise control. The functionality of each subsystem is carried out by a real-time task, and the tasks execute on two processors. The properties of all the tasks are summarized in Table 1. The utilization factor u_i of a task τ_i represents the fraction of CPU time demanded by task τ_i.*

The tasks are executing on two processors, $\mathcal{P}_1$ and $\mathcal{P}_2$. The total utilization factor of each processor is determined by the sum of the utilization factors of the corresponding tasks: the utilization factor of processor $\mathcal{P}_1$ equals 0.55*, while the utilization factor of processor $\mathcal{P}_2$ equals* 0.5*. Suppose that due to a failure of processor $\mathcal{P}_2$, all of the tasks must execute on the processor $\mathcal{P}_1$. The system*

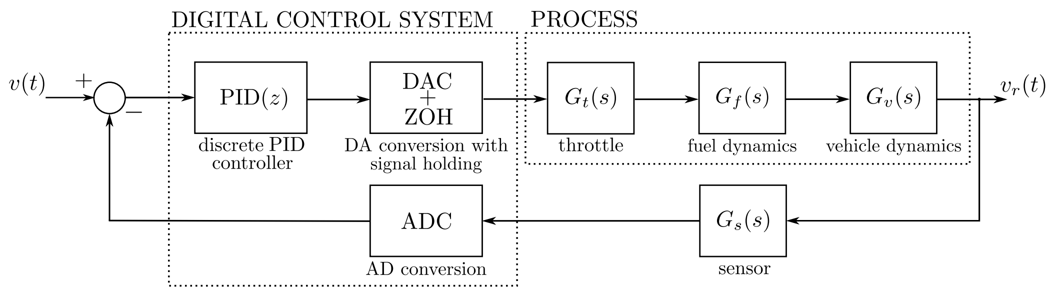


Figure 1: A computer-driven control system for car cruise control.

is now overloaded since total utilization factor equals: $u_1 + u_2 + u_3 + u_4 = 0.3+0.25+0.25+0.25 = 1.05$. *In that case, it is reasonable to adjust the policy for scheduling less critical tasks in order to reduce the system load and thus ensure that tasks of higher criticality, namely engine control, traction control, and cruise control, will meet their deadlines. We will analyze the behavior of the cruise control subsystem under overload and evaluate the possibility to decrease the system load by systematically rejecting some jobs of cruise control task, thus reducing its utilization.*

A simplified implementation of a control loop for car cruise control is depicted in Fig. 1. The regulator is modelled as a simple PID controller, while the components of the process, namely throttle, fuel dynamics, vehicle dynamics and sensor characteristic are modelled as first-order transfer functions. The parameters of these transfer functions are selected as in the example from [47]. We will analyze the response signal of the control loop with respect to timing constraints of the task that carries out the PID controller functionality. Suppose that the car is cruising at a constant speed of 50 *kilometers per hour. At* $t = 40s$, *the required speed is increased from* 50 *to* 55 *km/h. Note that this increase corresponds to a step signal with an amplitude of* 5. *We will analyze the response of the system* $v_r(t)$ *to the change in input speed. For reference, we simulated the behavior of the system in underload conditions, where all tasks completed before their deadlines. The output signal is shown in Fig. 2 by a solid line. The response signal obtained in this case corresponds to ideal behavior.*

In the second case, the system is overloaded and the total utilization factor equals 1.05. *Suppose that the scheduling algorithm guarantees that* 4 *out of* 5 *jobs of the cruise control task will be completed before their deadlines. Note that the ability to skip* 1 *in* 5 *jobs can reduce system load by exactly* 5%,

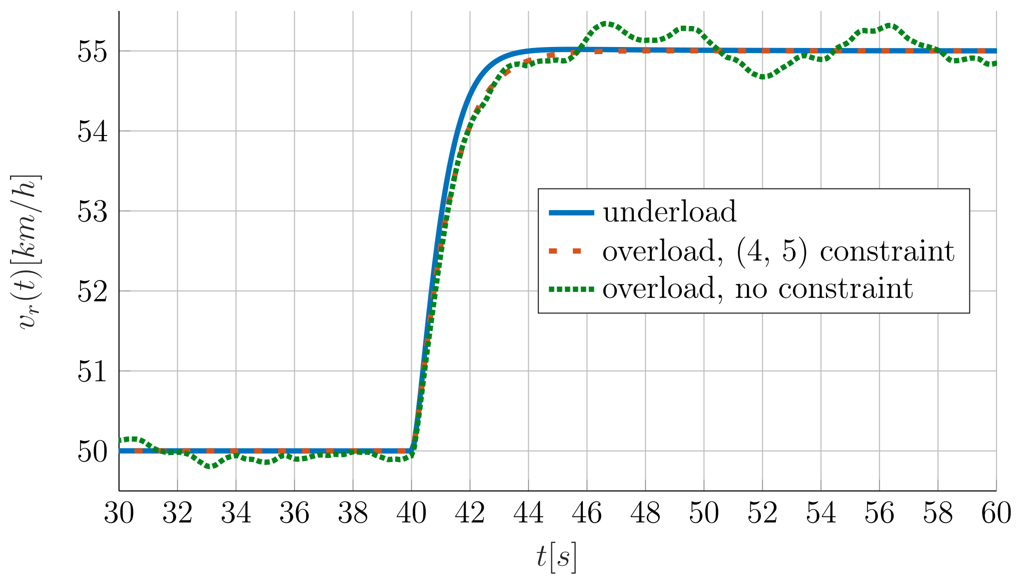


Figure 2: A comparison of the performance of the control system in underloaded and overloaded conditions. Two cases are considered for handling overloaded conditions: $(4, 5)$ weakly hard constraint and setting no constraints on deadline violations.

reducing the utilization of task τ_4 to $4/5 \cdot 0.25 = 0.2$. The total utilization factor is now $u_1 + u_2 + u_3 + u_4 = 0.3 + 0.25 + 0.25 + 0.2 = 1$. The obtained step response is shown in Fig. 2 by a dashed line. In this case, the performance is not significantly degraded and skipping a single job in a sequence of 5 jobs impacts the rise time of the step response only. More precisely, rise time is increased by 30% with respect to the response signal obtained in underloaded case.

Lastly, we will analyze a condition where the utilization factor is 1.05, but no constraints are set for the considered task. We assume that the scheduling algorithm ensures that jobs of tasks τ_1, τ_2, and τ_3 will complete before their deadlines, while the jobs of the task τ_4 will miss their deadlines with the probability of 20%. The response signal obtained in this case is shown in Fig. 2 by a dotted line. As in the previous case, the rise time of the response signal is increased, but the stability in steady state is significantly degraded.

In this example, we demonstrated how the consequences of overload may be mitigated by setting a constraint that enables discarding jobs according to a precisely defined pattern and thus reducing the effective utilization of the system. In this case, the deadlines are violated in a predefined and predictable

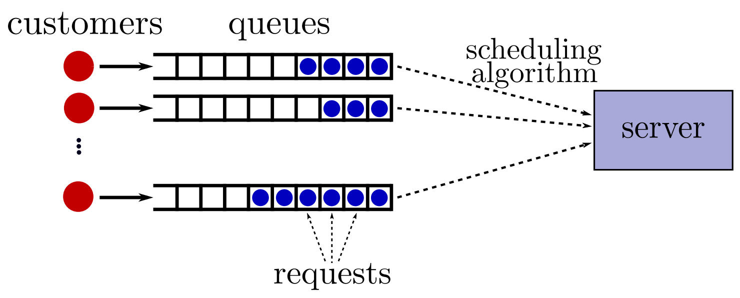


Figure 3: Multiple Input Queue Single Server (MIQSS) model.

manner and the performance of the system is slightly degraded, but its stability is not jeopardized.

2.2.2. Communication systems

Weakly hard real-time systems are also used for specifying the temporal constraints in communication systems that consist of messages from multiple streams that are transmitted through the same medium. The system model of this concept consists of streams of schedulable entities, referred to as customers, that must be scheduled on a single server [17]. In the literature, this model is referred to as MIQSS model (Multiple Input Queue Single Server) and it is illustated in Fig. 3. Each stream consists of a source and a queue. The server schedules a customer at the head of a queue according to its scheduling policy. The customers have a specified service time: the time required for serving the customer. Note that this parameter is equivalent to the execution time of tasks in real-time scheduling theory. The request of a customer must be fully serviced in a specified time interval which is determined by the deadline. The requests of the same customer are invoked with a specified inter-arrival time, which is equivalent to the period in classical real-time scheduling theory.

The performance of communication systems is usually evaluated by the quality of service metric: the ratio of fully serviced requests with respect to the total number of requests. In overloaded conditions, not all requests can be serviced and an increase in the utilization causes a degradation of quality of service. Although these systems are usually considered as soft real-time systems because violating the timing constraints does not cause catastrophic consequences, it can be useful to provide constraints which ensure that a certain quality of service will be achieved. Moreover, it can be important to provide a mechanism that guarantees not only the overall quality of service, but also the quality of service for each individual customer in the system. These functionalities can be achieved by including weakly hard constraints in the scheduling algorithm of the server.

In the described examples, we illustrated the motivation for the methods introduced in this work through practical use cases. In this paper, we focus on the optimization of the quality of service in task sets under the skip-over model, and we perform our analysis on synthetically generated task sets without targeting specific task sets from real-world applications such as control systems or multimedia systems.

3. System model and problem definition

3.1. System model

The real-time system model considered in this paper consists of a set $\mathcal{T}$ of N periodic tasks: $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. In classical real time scheduling theory, each task is characterized by means of three temporal parameters, a period T_i, deadline D_i, and worst-case execution time (WCET) C_i. In the system model considered in this work, the task deadlines are implicit, i.e., equal to the task periods, $D_i = T_i$. Each task releases an infinite sequence of instances, which are referred to as jobs J_{ij}. The dynamic parameters that characterize each job include remaining execution time $c_{ij}(t)$, absolute deadline $d_{ij}(t)$, and remaining time until deadline $\rho_{ij}(t)$. In the rest of the paper, we will refer to these parameters with annotations c_{ij}, d_{ij}, and ρ_{ij} for brevity. A utilization factor u_i of task τ_i corresponds to the fraction of CPU time demanded by task τ_i and it is computed as:

$$u_i = \frac{C_i}{T_i} \tag{1}$$

The total utilization factor of the task set $\mathcal{T}$ is computed as:

$$U(\mathcal{T}) = \sum_{i=1}^{N} u_i \tag{2}$$

In this work, we consider an extended real-time task model that includes weakly hard real-time constraints. A weakly hard real-time constraint defines the requirements on the pattern of missed and met deadlines for each task and in the literature it is denoted by λ_i [2]. Our work focuses on the skip-over task model which is based on a specific case of a weakly hard constraint denoted by $\left\langle{n \atop m}\right\rangle$ which specifies that in any window of m consecutive jobs, at least n jobs in a row must meet their deadlines. The constraints of the skip-over model are more conservative than $\left\langle{n \atop m}\right\rangle$ constraint. Specifically, in a window of m consecutive jobs, at least $m-1$ jobs must be timely completed. In skip-over model, this constraint is defined through the skip factor S_i: if a job of a task τ_i is skipped, at least $S_i - 1$ subsequent jobs of the same task must meet their deadlines. According to the skip-over model, the state of each job is described as either *red* or *blue*. A red job must complete within its deadline, whereas a blue job can be aborted (skipped) at any time. This requirement can be described by the following characteristics [48]:

- the distance between two consecutive skips must be at least S_i periods,
- if a blue job is skipped, the next $S_i - 1$ jobs are necessarily red,
- if a blue job completes successfully, the next job is also blue.

By definition in [3], the values of the skip factor can be in the interval $[2, \infty)$. Note that $S_i = \infty$ corresponds to a hard real-time system where no deadline violations are allowed. In our work, we consider task sets that also include soft real-time tasks where deadline violations are always tolerated. We specify soft real-time tasks by $S_i = 1$. The $(4, 5)$ constraint defined in example 2.1 can be represented in the skip-over model as $S_4 = 5$, while the skip factors for tasks τ_1, τ_2, τ_3 can be defined as $S_1 = S_2 = S_3 = \infty$.

The authors of the skip-over model have proven that the problem of determining the schedulability of a set of periodic skippable tasks is NP-hard [3]. However, they showed that the worst case for the feasibility of a task set consisting of skippable tasks is the *deeply red* condition: a condition when the first $S_i - 1$ released jobs of task τ_i are red. If a task set is schedulable under this condition, it is also schedulable in any other condition and therefore the feasibility tests for sets of skippable tasks are derived under the deeply red condition. A necessary schedulability condition was derived in [3], and a sufficient condition was introduced in [49]. As discussed in [3], the problem of making optimal use of skipping jobs is likely to be NP-hard. Therefore, the approaches for finding optimal job-skipping patterns, i.e., patterns that minimize the number of skipped jobs, are not adequate for an on-line implementation in real-time systems due to exponential time complexity of an optimal algorithm.

3.2. Optimizing quality of service of periodic task sets

The problem that is investigated in the literature in context of skip-over model is optimization of quality of service by choosing a scheduling algorithm $\mathcal{A}$ for periodically released jobs [4, 50]. Quality of service for a periodic task τ_i at time instant t for scheduling algorithm $\mathcal{A}$ is defined as:

$$q_i(t) = \frac{\text{number of releases} - \text{number of skips}}{\text{number of releases}} \tag{3}$$

The average quality of service (QoS) for a task set $\mathcal{T}$ can be defined as:

$$Q(\mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} q_i(t) \tag{4}$$

Note that the skipping factor is directly related to the minimal quality of service that can be achieved for a certain task. Skipping factor S_i guarantees that task τ_i will achieve a quality of service $q_i \geq (S_i - 1)/S_i$.

In this paper, our focus is on the optimization of the quality of service by constructing the scheduling algorithm $\mathcal{A}$ using the hyper-heuristic genetic programming approach. Prior to explaining the details of the approach, we revisit existing scheduling algorithms for skip-over model and discuss them in more detail.

3.3. Scheduling approaches

This section will present several algorithms known in the literature which are designed to reduce the system load by skipping some jobs in the task set. It is ensured that the remaining jobs will complete within their deadlines, while aiming to increase the total quality of service (QoS) of the task set.

3.3.1. RTO algorithm

The simplest skip-over algorithm is the *Red Tasks Only* algorithm (RTO). This algorithm systematically rejects all of the blue jobs, whereas the red jobs are scheduled according to *Earliest Deadline First* (EDF) algorithm. Note that the distance between each two skips in an RTO schedule is exactly S_i periods.

3.3.2. BWP Algorithm

The *Blue When Possible* (BWP) algorithm was introduced in [3]. This algorithm allows blue jobs to execute if there are no red jobs present in the system. If there is more than one blue job in the ready state, they can be dispatched according to several heuristics, including earliest deadline, latest deadline, a lookahead heuristic that verifies whether scheduling a certain job will introduce overhead in the future [3], etc. The red jobs are scheduled according to the EDF algorithm. The disadvantage of this approach is that it does not make full use of the slack time given by red jobs. More precisely, blue jobs that are currently running are immediately aborted if a red job is activated, regardless of whether the running blue job can be completed without jeopardizing the timely completion of the red job.

3.3.3. RLP Algorithm

The *Red Tasks as Late as Possible* algorithm, introduced in [4, 48], stimulates the execution of blue jobs by executing them at idle times of a preliminary schedule considering only the red jobs. Idle time refers to the time span in the schedule where none of the red jobs are in ready state. RLP algorithm is specified by the following characteristics [48, 50]. If there are no blue jobs in the system, red jobs are scheduled as soon as possible according to the EDF. Otherwise, red jobs are processed as late as possible and blue jobs are processed in the idle time of red jobs. It is worth noting that blue jobs can be scheduled in idle time by any scheduling algorithm.

The mechanism for determining the idle times of the red jobs relies on the *Earliest Deadline as Late as Possible* (EDL) algorithm. The EDL algorithm was introduced in [51] and it was initially designed for minimizing the response time of soft aperiodic tasks by dispatching them in the idle times of periodic tasks. In the context of the RLP algorithm, the EDL algorithm is used for determining the latest start time for red jobs in order to maximize the slack time for dispatching blue jobs. Moreover, once the idle intervals are known, it can be decided on-line whether it is feasible to accept blue jobs. Chetto and Chetto proved in [51] that the EDL algorithm is optimal in a sense that it guarantees the maximum idle time in a given interval. The start and length of the idle intervals in an EDL schedule can be specified by means of two vectors: a deadline vector and an idle time vector. A deadline vector contains the time instants at which the system is potentially in the idle state, i.e., the absolute deadlines of the red jobs. The lengths of the idle times corresponding to the deadlines are contained in the idle time vector. The methods for computing the deadline and idle time vectors are described in detail in [52]. Silly and Chetto proved in [52] that on-line computation of deadline and idle time vectors is required only upon arrival of a request when no other requests are already present in the system. In this context, the vectors need to be computed upon arrival of a blue job, when no blue jobs are present in the system, but also when a blue job completes because this causes a shift in the previously computed vectors. It is sufficient to track only the idle times from the first hyperperiod, i.e., the least common multiple of the task periods, since the schedule repeats itself in the subsequent hyperperiods. In the skip-over model, the hyperperiod must be computed by taking the skip factors into account:

$$\mathcal{H} = lcm(S_1 \cdot T_1, S_2 \cdot T_2, ..., S_n \cdot T_n). \tag{5}$$

It is shown in [52] that the computational complexity of the deadline and idle time vectors is $O(\lfloor R/p \rfloor N)$, where R and p are the longest deadline and the shortest period of active jobs, respectively. Since on-line computation of the deadline and idle time vectors introduces additional overhead, it is not suitable for implementation on hardware platforms that are typically used in embedded systems due to limited resources.

An interesting observation about both BWP and RLP algorithms is the ability to arbitrarily choose a heuristic for scheduling of blue jobs without jeopardizing the feasibility of the system with regard to skipping factors. Intuitively, the quality of service can be optimized only by choosing different heuristics for scheduling blue jobs, which are executed in slack time of red jobs in case of the BWP algorithm or in the idle time of red jobs scheduled as late as possible in case of the RLP algorithm.

3.4. Problem definition

Formally, the problem that we solve in this paper is the maximization of the quality of service:

$$\text{maximize} \quad Q(\mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} q_i(t) \tag{6}$$

for any task set $\mathcal{T}$ for which the sufficient schedulability condition is satisfied and thus an algorithm that produces a feasible schedule exists. Less formally, the above problem can be described as: *Find a heuristic from the space of heuristics which optimizes the observed metric across a set of problem* *instances.* In this context, problem instances are different sets of periodic skippable task sets and the considered metric is the quality of service. These heuristics in our proposed approach are generated using the hyper-heuristic genetic programming approach which will be described in detail in Section 4. In this work, the heuristics are considered as priority functions that are integrated into scheduling meta-algorithms (BWP, RLP) or they act as standalone scheduling algorithms. In the following text, we provide a formal definition of the priority function.

3.5. Priority functions

The heuristics evolved by genetic programming correspond to priority functions. A priority function is an expression for computing the priority of each active job, as defined below:

Definition 3.1. Priority function. Priority function $\pi_{ij}(t)$ assigns a value to a job $\mathcal{J}_{ij}$ of task τ_i at some time instant t. The lower the assigned value, the higher is the priority.

The priority function enables the definition of scheduling policies (heuristics) in a relatively simple manner. For instance, the EDF scheduling policy can be simply defined as $\pi_{ij}(t) = d_{ij}(t)$, i.e., the lower is the deadline, the higher is the priority.

3.6. Motivational example

In the following text, we will provide an example of building schedules for an overloaded task set based on the algorithms described in the previous subsection. We will demonstrate how system load can be reduced by skipping a certain number of jobs and we will compare the presented approaches with respect to achieved quality of service. Moreover, we will consider two approaches that employ a custom heuristic for scheduling skippable jobs: in the first case, the custom heuristic is used for scheduling blue jobs in the context of RLP algorithm (GP-QoS-MA approach), whereas in the second case the heuristic acts as a stadalone scheduling algorithm (GP-QoS-S approach). We will demonstrate how custom scheduling heuristics can maximize the quality of service of the given task set.

Example 3.1. *Consider a task set given in Table 2. Since the utilization factor of the set $U(\mathcal{T}) > 1$, the task set is overloaded:*

$$U(\mathcal{T}) = \sum_{i=1}^{N} \frac{C_i}{T_i} = \frac{2}{8} + \frac{4}{8} + \frac{3}{6} = 1.25. \tag{7}$$

Table 2: Task set parameters for Example 3.1.

Task	C_i	$P_i = D_i$	S_i
τ_1	2	8	1
τ_2	4	8	2
τ_3	3	6	2

We aim to find a scheduling heuristic which provides a schedule with minimal number of skipped jobs—a heuristic with the maximal QoS. First, we observe a schedule provided by RTO algorithm, depicted in Fig. 4. The activation times of the jobs are denoted by upwards arrows, while deadlines are denoted by downwards arrows. The total number of released jobs equals 20. Since RTO systematically rejects every blue job, there are no completed jobs of task τ_1 *and there are 7 completed jobs of tasks* τ_2 *and* τ_3*. The total quality of service equals 0.35.*

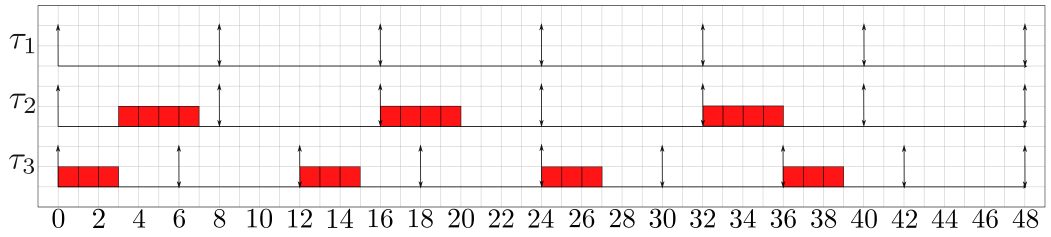


Figure 4: Schedule generated according to the RTO algorithm for the task set given in Table 2.

Fig. 5 shows a schedule produced by the BWP algorithm. Note that red jobs always have a higher priority than blue jobs. The blue jobs that are skipped are denoted in the schedule by a hatched pattern. In this case, 6 blue jobs were able to execute to completion and the total quality of service is now increased to 0.6.

Next, we will observe a schedule produced by the RLP algorithm. Blue jobs now have the same priority as red jobs, and idle time given by red jobs is maximally utilized. In this case, blue jobs are scheduled according to the EDF algorithm. The obtained schedule is shown in Fig. 6. The total quality of service obtained in this case is 0.6. Although RLP usually performs slightly better than BWP in terms of QoS [4], for this particular task set BWP and RLP yield the same QoS.

We will now analyze the possibility for using an alternate heuristic for

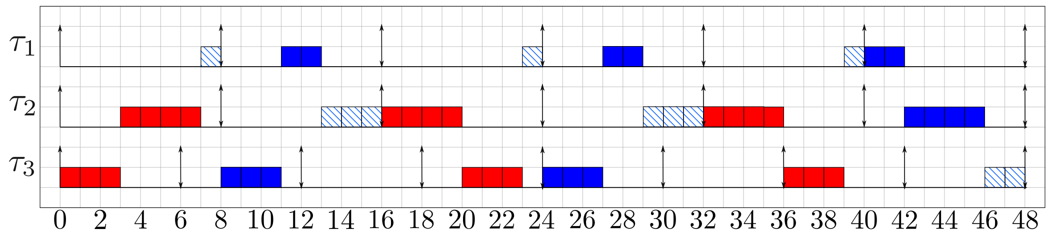


Figure 5: Schedule generated according to the BWP algorithm for the task set given in Table 2.

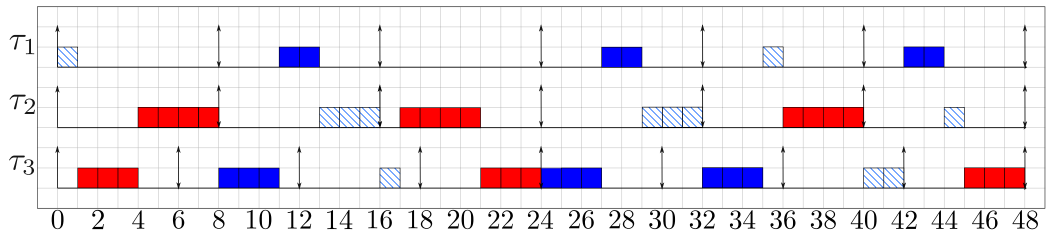


Figure 6: Schedule generated according to the RLP algorithm for the task set given in Table 2, with blue jobs scheduled according to EDF.

scheduling blue jobs in combination with RLP algorithm which would minimize the total number of skips. We will consider assigning the priority values to each blue job according to the following priority function:

$$\pi_{ij}(t) = \min\left(\max(\rho_{ij}, S_i), \frac{S_i}{\rho_{ij}}\right) \tag{8}$$

where parameter ρ_{ij} denotes the remaining time until deadline for job $\mathcal{J}_{ij}$ and S_i is the skip factor for task τ_i. The division operator is implemented as protected division: if the denominator is zero, the operator returns 1. *It is important to emphasize that jobs with a lower value of priority function $\pi_{ij}(t)$ execute first, i.e., the lower the value of priority function $\pi_{ij}(t)$, the higher the priority.*

The obtained schedule is shown in Fig. 7 and it can be seen that the total quality of service is now increased to 0.7. A table that denotes the calculated priority values of all active jobs is given in Appendix A.

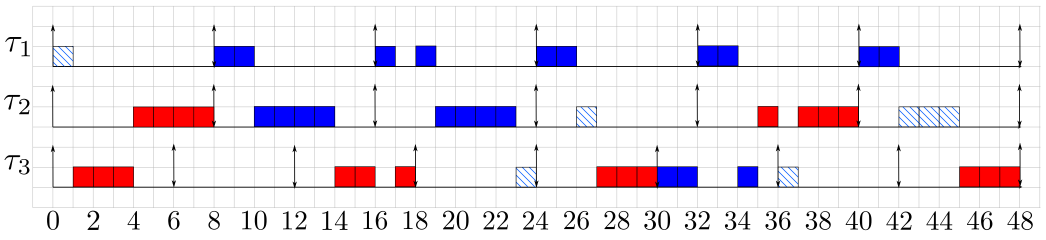


Figure 7: Schedule generated according to the RLP algorithm for the task set given in Table 2, with blue jobs scheduled according to priority function given in (8).

Finally, we consider a scheduling heuristic that acts as a standalone job-skipping algorithm, i.e., the priorities of both red and blue jobs are assigned according to the following expression:

$$\pi_{ij}(t) = \max\left(\frac{\rho_{ij}}{S_i}, \frac{C_i}{\sigma_{ij}}\right) \tag{9}$$

where parameter σ_{ij} denotes the state of job $\mathcal{J}_{ij}$:

$$\sigma_{ij} = \begin{cases} 0 & \textit{if job is red,} \\ 1 & \textit{otherwise.} \end{cases} \tag{10}$$

The obtained schedule is shown in Fig. 8. We can note that there are no violations of the skip factor, even though the scheduling heuristic does not provide a mechanism that guarantees that the constraint given by skip factor will be fulfilled. Moreover, the total quality of service is increased to 0.75. The priorities of active jobs are given in Appendix A, Table A.5.

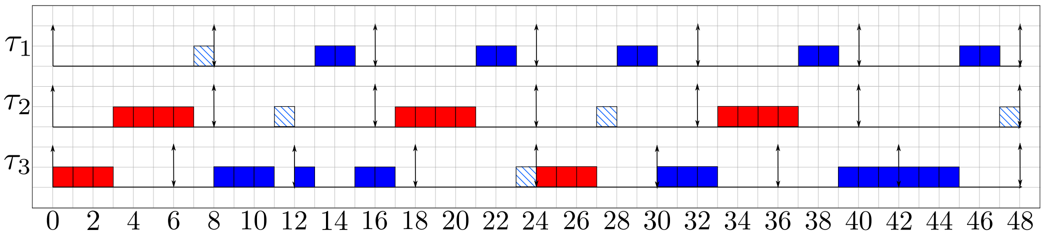


Figure 8: Schedule for the task set given in Table 2, obtained by standalone priority function given in (9).

The latter examples illustrate the motivation for including custom scheduling policies into algorithms that guarantee the satisfaction of weakly hard constraints corresponding to the skip-over task model. Moreover, it is worth considering the optimization techniques for finding the adequate heuristic for scheduling blue jobs, which can reduce the total number of skipped jobs and thus increase the total QoS. We also showed how scheduling heuristics generated by optimization techniques specifically for a given task set can be used as standalone job-skipping algorithms that maximize the QoS, while ensuring that the constraint defined through skip factor will be fulfilled.

4. Methodology: evolving scheduling heuristics with genetic programming

This section will describe the method for employing genetic programming for evolving scheduling heuristics. Genetic programming is suitable for searching the space of scheduling heuristics that provide schedules for an arbitrary scheduling environment [6], while aiming to find a heuristic that yields the best results considering the requirements of the system model. We evaluate the best heuristic on a large number of task sets for validation and compare the obtained results to state-of-the art algorithms for task scheduling under the skip-over model, namely RTO, BWP, and RLP. The setup for experimental evaluation and the obtained results will be presented in Section 5.

We use genetic programming for finding the priority function which best suits the given meta-algorithm and user demands regarding the system performance. A single priority function, i.e., solution to the optimization problem is referred to as an individual, while a set of individuals is referred to as a population.

The priority functions evolved by genetic programming are represented as tree structures that consist of function and terminal nodes [7]. Function nodes correspond to standard arithmetical operators, while the terminal nodes represent static or dynamic properties of jobs. An overview of the terminal and function nodes used in this work is given in Table 3.

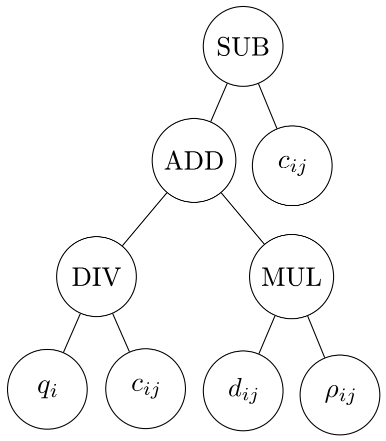


Figure 9: Example of a priority function represented with a tree structure.

Table 3: Description of function and terminal nodes.

Function name	**Description**
ADD, SUB, MUL	addition, subtraction and multiplication of child nodes
DIV	protected division (returns 1 if second argument is zero)
MAX	returns maximum value of two arguments
MIN	returns minimum value of two arguments

Terminal name	**Description**
C_i	execution time of task τ_i
T_i	period of task τ_i
S_i	skip factor of task τ_i
c_{ij}	remaining execution time of active job $\mathcal{J}_{ij}$
d_{ij}	absolute deadline of active job $\mathcal{J}_{ij}$
ρ_{ij}	remaining time to deadline of active job $\mathcal{J}_{ij}$
q_i	current quality of service of task τ_i
σ_{ij}	job state: 0 if job is red, 1 otherwise, used for GP-QoS-S only

The priority value is calculated by traversing the tree and evaluating the corresponding expression. An example of a tree structure which represents a scheduling heuristic is depicted in Fig. 9. The tree shown in the example corresponds to the priority function $\pi_i(t) = \frac{q_i}{c_{ij}} + d_{ij}\rho_{ij} - c_{ij}$. In the approaches presented in this work, priority functions are encapsulated into meta-algorithms, which define the manner of scheduling jobs at a certain time instant depending on their priorities and other system constraints.

The proposed approach is depicted in Fig. 10. The inputs of the optimization process include the parameters of the scheduling environment and the genetic algorithm. The scheduling environment parameters determine how the task sets will be generated and they include the number of tasks in the set, the maximum period of the tasks in the set, the distribution of the utilization factors per task set, and the configuration of skip factors per task. The task sets for training and validation were generated using the UUnifast algorithm [53]. This algorithm is commonly used in research papers that deal with analyzing the performance of scheduling algorithms as it allows the synthetization of a large number of task sets with a given utilization factor. The parameters of the genetic algorithm include the population size, the termination condition, configuration of mutation and crossover operators, etc. The exact configuration of the environment and genetic algorithm parameters will be described in Section 5. The output of the process is the best heuristic evolved by the genetic algorithm, whose performance is evaluated on the task sets for validation. In the validation process, we examined the performance of the best heuristic with respect to several parameters of the task sets, namely utilization factor, maximum task period, number of tasks in the set, and skip factor. The analysis of the average performance of the heuristics in comparison with other job-skipping algorithms is given in Section 5. A thorough statistical analysis of the obtained results is given in Appendix D.

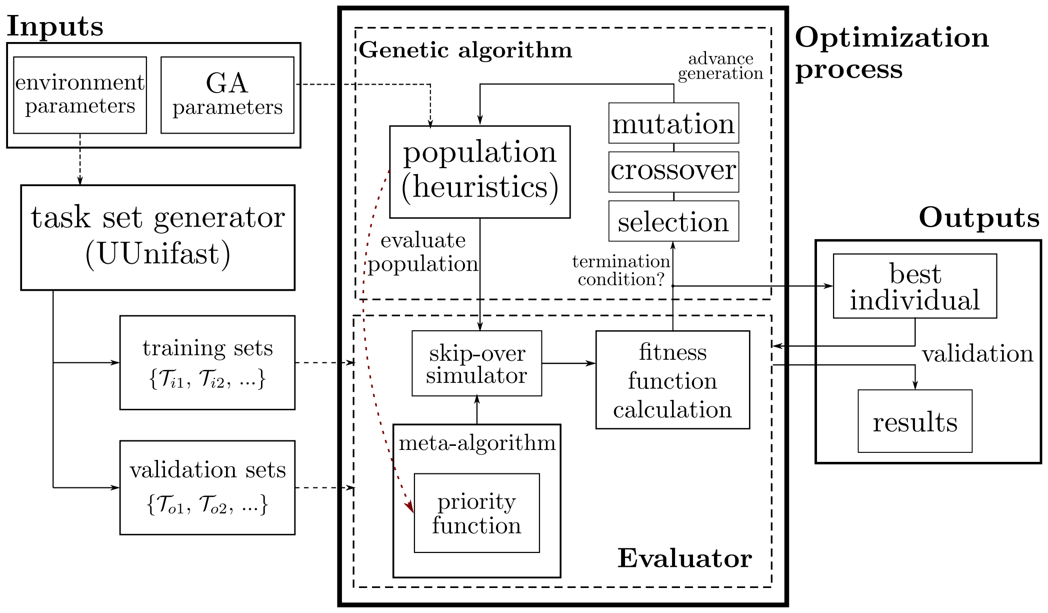


Figure 10: Genetic programming approach for design of heuristics for scheduling blue jobs.

The optimization process can be divided into two components: an evaluator and a genetic algorithm. We used a genetic algorithm implemented in the *Evolutionary Computation Framework* (ECF) [54]. The framework implements a steady-state variant of the genetic algorithm. In every step of the algorithm, each individual is evaluated, i.e., it is assigned a fitness value according to its performance in solving the problem of interest. After the evaluation of the individuals in the population, the genetic algorithm performs selection of two individuals which will represent parents. Then, the operators of crossover and mutation are applied to form a new individual which is referred to as the offspring. In the genetic programming approach that we use in this work, crossover and mutation operators are implemented as sub-tree crossover and sub-tree mutation as described in [55]. The offspring then replaces the worst individual in the population. As the evolution process continues, the individuals converge towards the optimal heuristic. This process repeats until the termination condition is fulfilled. The termination condition can be defined through various events or states, but the most commonly used include:

- evolution reached a maximum number of generations,
- evolved individuals reached the target fitness value,
- maximum number of consecutive generations without improvement exceeded.

In this case, the individuals are evolved through a predefined number of generations, which means that the evolution process is stopped once the maximum number of generations is reached. When the termination condition is fulfilled, the algorithm returns the best individual in the population.

The evaluator assigns fitness values to individuals by simulating the execution of task sets in a real-time environment. For the purpose of this work, we implemented an evaluator that corresponds to the optimization problem addressed in this work and a simulator of a real-time environment corresponding to the skip-over model. In the simulation, the jobs are scheduled according to a given meta-algorithm and the individual that is being evaluated is integrated in the meta-algorithm as a priority function.

The execution of the task sets in the simulation is configured as shown in Example 3.1: the jobs are released periodically with no initial offset, the overload conditions are simulated by executing each job for its worst-case execution time (WCET). Moreover, we simulate the execution of the tasks under deeply red condition— the first $S_i - 1$ jobs of each task are in the red state. At every time instant, the highest-priority job is selected as the running job according to the considered meta-algorithm. The meta-algorithms are described in more detail in Subsection 4.1. The execution of the task set is simulated in the first hyperperiod only. When the simulation time expires, each individual is assigned a fitness value based on its performance in the simulation. The computation of fitness functions will be described in more detail in Subsection 4.2. A thorough description of the simulator can be found in Appendix B.

The result of the optimization process is the best individual, i.e., the heuristic that obtained the best performance in the evaluations. The best individual is then evaluated on the task sets for validation.

4.1. Scheduling meta-algorithms

In the GP-QoS-MA approach, we integrate the priority functions generated by genetic programming into conventional job-skipping algorithms, namely BWP and RLP. In the scheduling environment that corresponds to these algorithms, priority functions are used for scheduling blue jobs and they are incorporated into different meta-algorithms, depending on the scheduling technique that is being used (BWP or RLP). The outline of the meta-algorithm corresponding to the BWP approach is shown in Alg. 1. If there is any active red job, the scheduler always dispatches a red job, regardless of the priority of blue jobs. On the other hand, RLP approach stimulates the execution of blue jobs by determining whether the current time instant belongs to an idle interval of red jobs. The meta-algorithm for the RLP approach is given in Alg. 2.

In the GP-QoS-S approach, the priority functions are used as standalone algorithms and the priorities are assigned in the same manner for red and blue jobs. Therefore, the meta-algorithm consists of priority assignment and dispatching only, as shown in 3.

Algorithm 1 Meta-algorithm for BWP scheduling

```
1: if there are active red jobs then
2:     schedule red job with the earliest deadline
3: else
4:     calculate priorities $\pi_{ij}(t)$ for every blue job
5:     schedule blue job with highest priority
```

Algorithm 2 Meta-algorithm for RLP scheduling

```
1: if $t$ belongs to idle interval then
2:     calculate priorities $\pi_{ij}(t)$ for every blue job
3:     schedule blue job with highest priority
4: else
5:     schedule red job with the earliest deadline as late as possible
```

Algorithm 3 Meta-algorithm for GP-QoS-S approach

```
1: calculate priorities $\pi_{ij}(t)$ for all jobs
2: schedule job with the highest priority
```

4.2. Fitness functions

The fitness of the individuals is assigned through an evaluator, which simulates the behavior of the input task sets for each scheduling heuristic in the population.

In the GP-QoS-MA approach, the fitness of an individual depends on the corresponding priority function that is used for best-effort scheduling of blue jobs. The fitness value is directly related to the quality of service obtained by the generated schedule and it is computed as described in Equation 6.

On the other hand, in the GP-QoS-S approach, the same heuristic is used for scheduling both red and blue jobs. This approach does not provide a mechanism which guarantees that no red jobs will be skipped. However, by using an adequate fitness function, the heuristic can ensure that no violations of the skip factor will occur for a given task set. Thus, in the GP-QoS-S approach, we modified the QoS fitness function in a way that the violations of the skip factor constraint are penalized. Specifically, this fitness function, denoted by $F(\mathcal{T})$, subtracts the number of skipped red jobs from the QoS value and thus yields a negative QoS for a heuristic that allows the violations of the skip factor. The value of fitness function $F(\mathcal{T})$ is computed as:

$$F(\mathcal{T}) = Q(\mathcal{T}) - \sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathcal{V}(i,j) \tag{11}$$

where M_i is the number of released jobs of task τ_i in the first hyperperiod, and $\mathcal{V}(i,j)$ is a function that determines whether job $\mathcal{J}_{ij}$ has violated the skip factor S_i, i.e., job $\mathcal{J}_{ij}$ is in red state and has skipped its deadline:

$$\mathcal{V}(i,j) = \begin{cases} 1 & \text{if job } \mathcal{J}_{ij} \text{ is in red state and skipped,} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

It is important to emphasize that there is a difference in the evolution process for GP-QoS-MA and GP-QoS-S approaches. In the GP-QoS-MA approach, a single priority function is applied to all of the training task sets $(\mathcal{T}_{i1}, \mathcal{T}_{i2}...)$. The idea of the GP-QoS-S approach is to evolve a scheduling heuristic specifically for a given task set and therefore we perform a separate evolution process for every training task set. This approach thus yields multiple scheduling heuristics - one for each training task set $\mathcal{T}_{ij}$.

5. Experimental evaluation

In this section, we evaluate the effectiveness of using heuristics evolved by genetic programming for scheduling jobs in a weakly hard real-time environment. For implementation of the genetic programming procedure, we used the *Evolutionary Computation Framework* (ECF). The current version of the framework is available from [54]. The performance of our approach is evaluated with respect to several parameters, namely the utilization of task sets $U(\mathcal{T})$, the maximum period in the task set $T_{\max}$, the number of tasks in the set N, and maximum skip factor in the set $S_{\max}$. We compare the performance of the heuristics with standard implementations of RTO, BWP and RLP.

5.1. Task set generation

As mentioned in the previous section, we used the UUnifast algorithm for generating the task sets for training and validation. The methods for selecting the number and the parameters of the sets for training and validation will be described in Subsections 5.2 and 5.3. We can describe the process of synthesizing the task sets as follows:

- the total utilization factor of the task set $U(\mathcal{T})$ is chosen,
- the utilization of each task in the set u_i is computed according to the UUnifast algorithm,
- period value T_i for each task is chosen as a random variable with uniform distribution from the interval given with minimum and maximum period values ($T_{\min}$ and $T_{\max}$),

- the WCET for each task is calculated from the values u_i and T_i.

Using a constant skip factor for every task simplifies the scheduling problem and does not yield accurate results. Upon creation of every task set, it is validated that the set is schedulable according to the sufficient schedulability condition, which is defined in [49]. In order to decrease the runtime of the simulations, we set an upper limit to the hyperperiod of the tasks to 10000. In each of our experiments, we varied one of the parameters, while the others were set to their default values. The default values of the task parameters are as follows:

- utilization factor $U(\mathcal{T}) = 1.2$,
- the number of tasks in the task set $N = 6$,
- maximum allowed period $T_{\max} = 500$,
- maximum skip factor in the task set $S_{\max} = 6$.

The skip factor for each task is generated as a random variable with uniform distribution from the interval $[1, S_{\max}]$.

5.2. Selection of genetic programming parameters

In order to determine the specifications of the environment for evolution of scheduling heuristics, we explored several crucial parameters, namely the number of generations, the genotype size, and the number of individuals in the population. We analyzed the impact of the considered parameters and determined the parameter values that are most suitable for our study:

- stopping criterion: 15 generations reached,
- population size: 100,
- maximum depth of the tree genotype: 4.

A thorough analysis of the parameters is given in Appendix C.

We also observed the influence of the parameters for crossover, mutation, and selection operations and concluded that the impact of these parameters is much smaller than the parameters such as tree depth which we analyzed in more detail. Therefore, for these parameters typical values were chosen. In the evaluation, we used steady-state tournament as the selection operator, with size of the tournament set to 3. We used a generic approach for crossover and mutation in genetic programming, which includes sub-tree swapping and sub-tree mutation. Typically, the mutation of the offspring occurs with a given probability and in this case we set the probability of the mutation to 0.3.

5.3. Experimental results

In this section, we analyze the results obtained by the evolved scheduling heuristics for GP-QoS-MA and GP-QoS-S approaches and compare their performance with standard implementations of RLP, BWP, and RTO algorithms. Additional experiments that verify the statistical significance of the obtained results are described and discussed in Appendix D. It is important to emphasize that RLP and BWP are state-of-the-art algorithms for scheduling tasks under skip-over model and our research is the first to present methods that outperform these algorithms.

In our evaluation, we provide a comparison with all of the existing algorithms that address the skip-over model. Moreover, we analyze the performance of the proposed approaches in comparison with the EDF algorithm and the *Shortest Remaining Time First* (SRTF) algorithm. The SRTF algorithm typically brings over-the-average performance with regard to QoS since it gives the highest priority to the jobs that require the lowest amount of CPU time. However, EDF and SRTF provide no mechanisms for ensuring that the weakly hard constraints will be fulfilled.

In the experiments, we measured the performance of a priority function that corresponds to the best individual obtained by genetic programming. The performance of the best individual is evaluated over a larger amount of task sets that were not included in the training process. More precisely, we generated 100 times more task sets for validation to demonstrate that the evolved heuristic has learned generalized behavior. We validated the priority functions with respect to several parameters, namely the utilization of the task sets, the number of tasks in the system, the maximum allowed period of the tasks in the task set, and two different principles of skip factor assignment. Besides analyzing the performance regarding the objective function, i.e., QoS, we also discuss the performance of the presented approaches with regard to a metric that corresponds to the number of violations of the skip factor. For each task τ_i, this metric is computed as a ratio of skipped red jobs in the total number of released jobs:

$$n_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \mathcal{V}(i, j) \tag{13}$$

where M_i is the number of released jobs of tasks τ_i in the first hyperperiod and $\mathcal{V}(i, j)$ is computed as described in (12).

In our evaluations, we consider an average value of this metric for the given task set and it is computed as:

$$N(\mathcal{T}) = \frac{1}{N} \sum_{i=1}^{N} n_i \tag{14}$$

5.3.1. Evaluation of the GP-QoS-MA approach

Firstly, we compare the performance of the GP-QoS-MA approach with regard to the utilization factor of the task sets $U(\mathcal{T})$. We varied the utilization factor in the interval $[0.9, 1.4]$ with an increment of 0.05. For training process, we generated 100 task sets per each utilization factor which results in a total of $((1.5 - 0.9)/0.05 + 1) \cdot 100 = 1300$ task sets for training. In the validation process, we generated 10000 task sets per each utilization factor, thus the total number of task sets for validation equals 130000. We expect that the increase in utilization factor will decrease the average quality of service. Fig. 11 shows the performance of the evolved heuristics and conventional algorithms (RTO, BWP, RLP) with regard to average quality of service at the end of hyperperiod. The heuristics obtained by the approach described in this paper are denoted by BWP+GP-QoS-MA and RLP+GP-QoS-MA and it can be seen that they dominate conventional job-skipping algorithms in overloaded conditions. If we observe the rest of the algorithms, we can see that RTO yields the lowest QoS, which is expected. BWP and RLP perform similarly in terms of average QoS, both in underloaded and overloaded conditions. It is interesting to notice that the BWP+GP-QoS-MA approach performs worse than standard BWP when $U(\mathcal{T}) < 1$. This is due to the fact that standard BWP employs the EDF algorithm for scheduling blue jobs, which is optimal for $U(\mathcal{T}) < 1$. Consequently, all jobs released after resolving the deeply-red condition are blue. On the other hand, an arbitrary heuristic for scheduling blue jobs in the BWP+GP-QoS-MA approach may not be optimal and some of the jobs may be skipped. This causes the activation of red jobs and the remaining blue jobs are aborted until red jobs are completed. This behavior

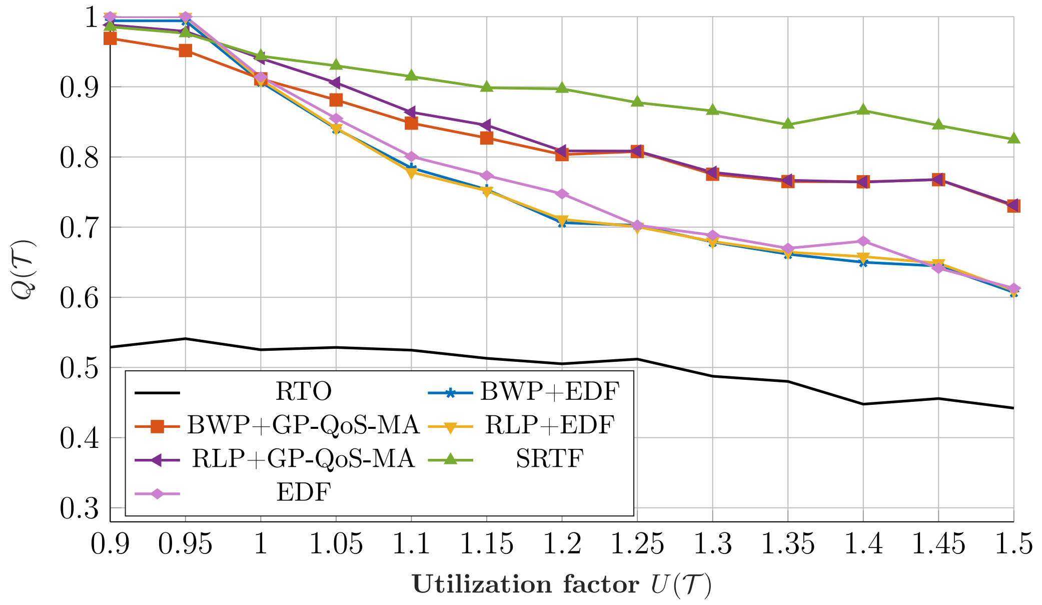


Figure 11: Average QoS with respect to the utilization factor $U(\mathcal{T})$.

is not as emphasized in the RLP+GP-QoS-MA approach because in that case, upon skipping a job, the activated red job is postponed to be executed as late as possible, giving the remaining blue jobs maximum slack time for execution.

For completeness, we simulated the behavior of scheduling algorithms EDF and SRTF on the validation task sets. We notice that EDF performs similarly as RLP and BWP under overloaded conditions and SRTF yields significantly higher values of average QoS when $U(\mathcal{T}) > 1$. However, SRTF brings an artificial improvement of the average QoS since it penalises the most time-consuming jobs. Moreover, EDF and SRTF do not guarantee the satisfaction of the weakly hard constraint given by the skip factor, which is illustrated in Fig. 12. The figure shows the performance of the algorithms with respect to the frequency of the violations of the weakly hard constraint, i.e., the $N(\mathcal{T})$ metric. The algorithms RTO, BWP, and RLP, both conventional and combined with evolved heuristics, guarantee the satisfaction of weakly hard constraint defined by the skip factor. Therefore, the graphs RTO, BWP+EDF, BWP+GP-QoS-MA, RLP+EDF, and RLP+GP-QoS-MA are on the x-axis, i.e., $N(\mathcal{T}) = 0$. SRTF and EDF, however, allow the violations of the constraint and the number of violations escalates when the utilization

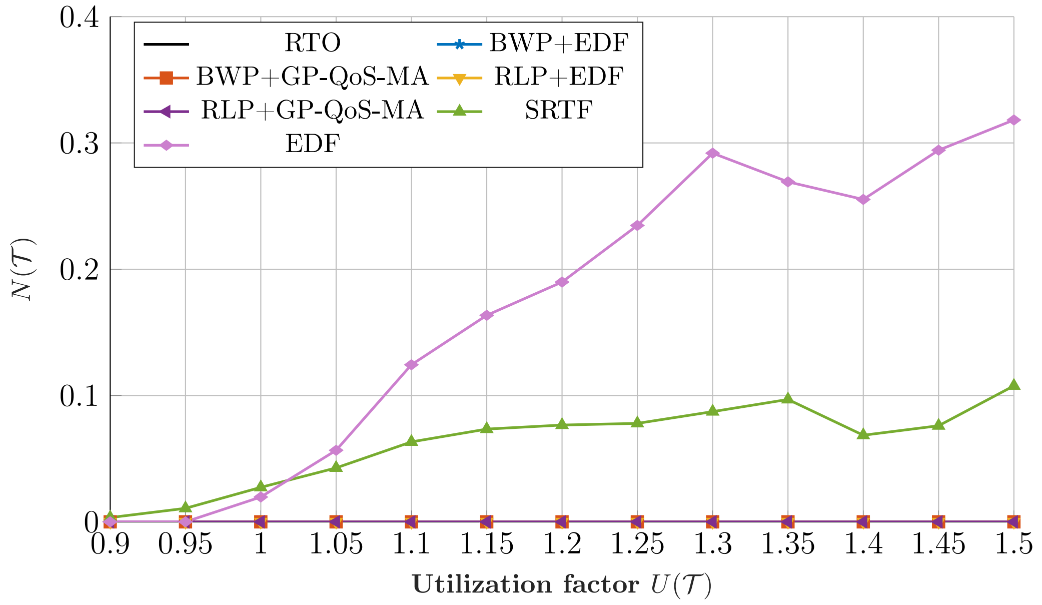


Figure 12: Average ratio of skip factor violations with respect to the utilization factor $U(\mathcal{T})$.

factor is increased.

Further, we analyze the performance of the described approach with respect to different values of maximal allowed period in the task set. We varied the maximal period values from $T_{max} = 100$ to $T_{max} = 1000$ with increment of $T_\delta = 100$. This corresponds to a total of $((1000 - 100)/100 + 1) \cdot 100 = 1000$ sets for training and $((1000-100)/100+1) \cdot 10000 = 100000$ sets for validation. In this experiment, the utilization factor is fixed to 1.2. The number of tasks in the task sets is 6, and the skip factors for the tasks are generated randomly from the interval $[1, 6]$. Fig. 13 shows that the heuristics generated by genetic programming generalize well when the maximum task period T_{max} is varied and they perform better than the conventional job skipping algorithms in terms of quality of service. The evolved heuristics yield expected results in terms of weakly hard constraint satisfaction, which is shown in Fig. 14.

Next, we compare the performance of the described approaches with regard to different number of tasks in a task set. We generate task set sizes for each integer N, $4 \leq n \leq 8$ which yields a total of $5 \cdot 100 = 500$ sets for training and $5 \cdot 10000 = 50000$ sets for validation. As in the previous experiment, the utilization factor of the task sets is fixed to 1.2. Fig. 15 shows that the

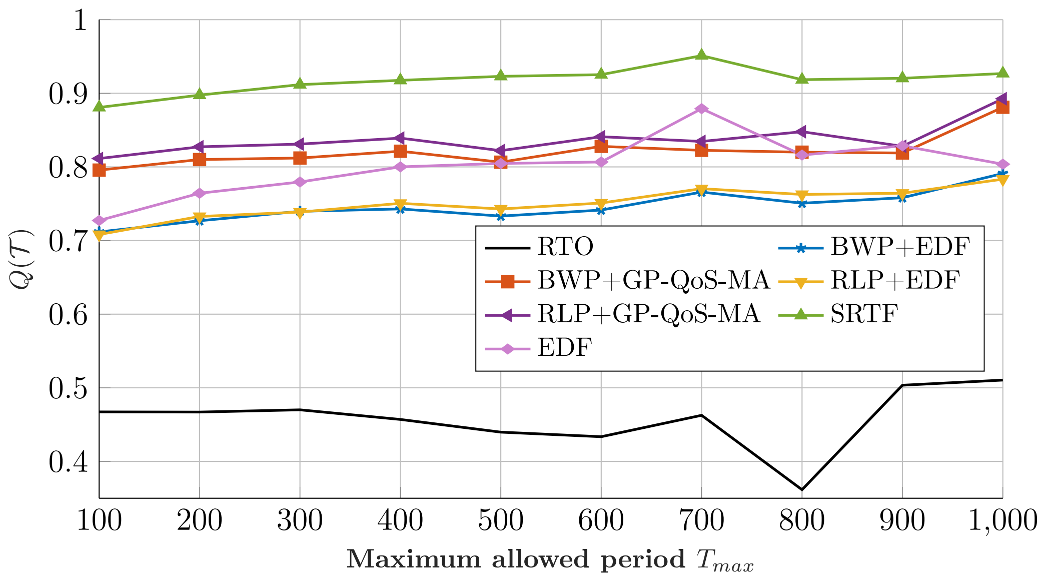


Figure 13: Average QoS with respect to the maximum allowed period T_{max}.

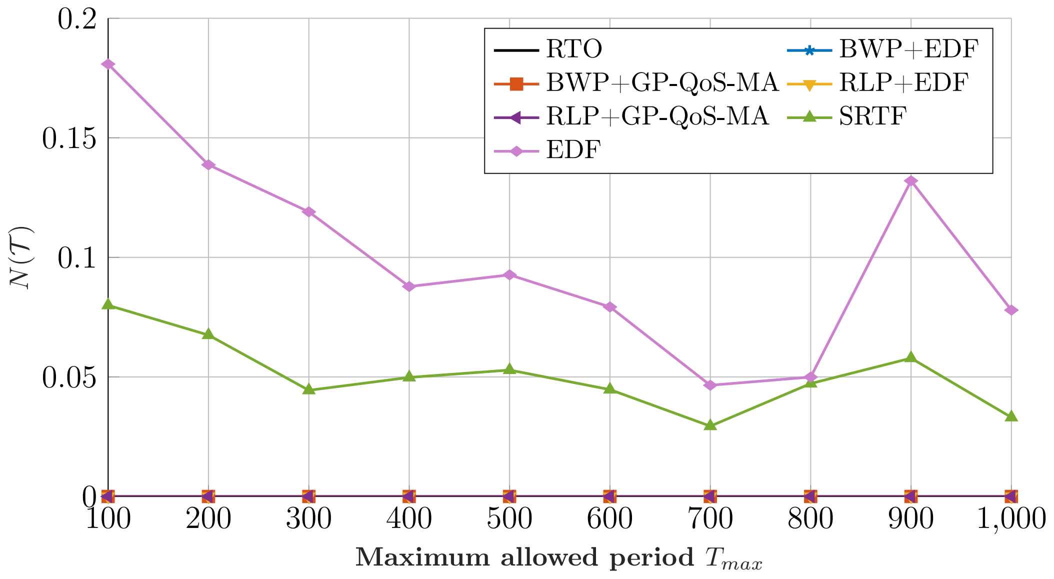


Figure 14: Average ratio of skip factor violations with respect to the maximum allowed period T_{max}.

proposed genetic programming approach generalizes well and achieves the best performance in overload conditions, regardless of the size of the task sets. Once again, the $N(\mathcal{T})$ metric is greater than zero only for task sets scheduled by EDF and SRTF, which is shown in Fig. 16.

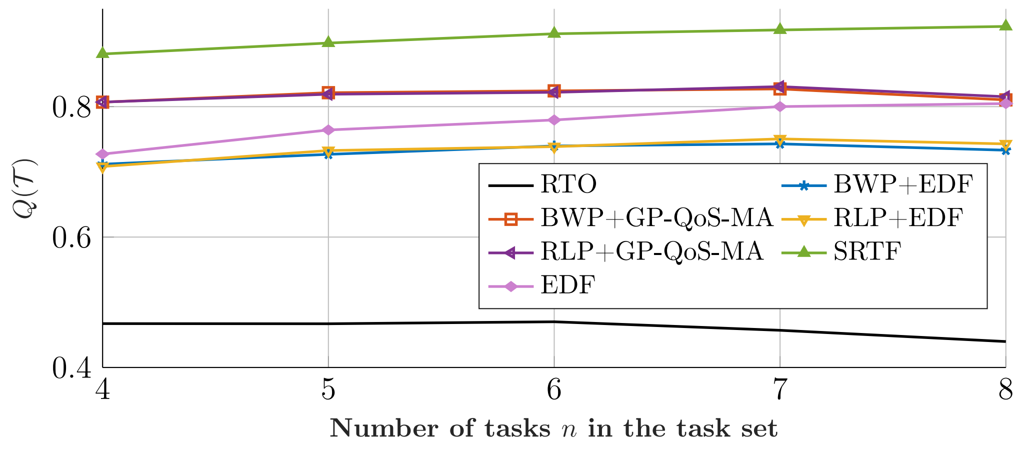


Figure 15: Average QoS with respect to the number of tasks N in the task set.

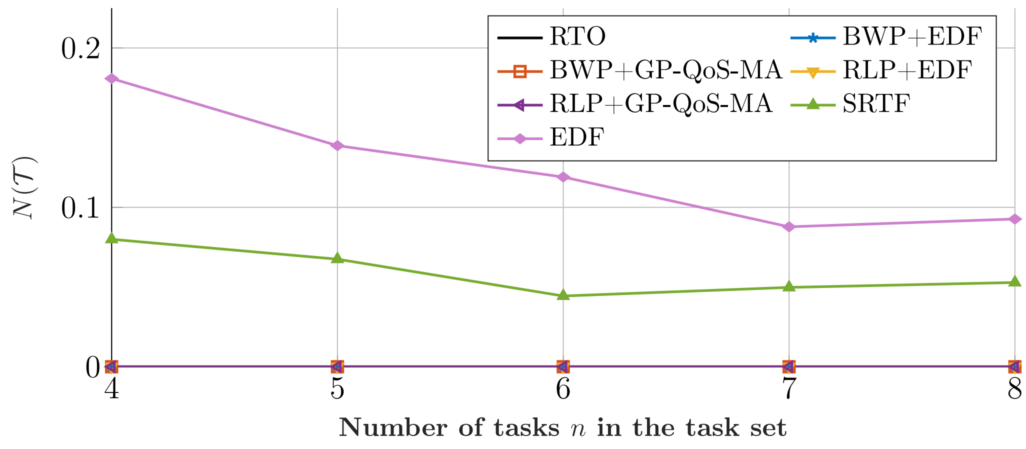


Figure 16: Average ratio of skip factor violations with respect to the number of tasks n in the task set.

In the last experiments, we examine the achieved quality of service with respect to the skip factor of the tasks. We analyzed two different configurations. First, we generated skip factors for each task randomly, while varying the upper bound on the generated values. The lower bound for skip factors of the tasks is set to 1, while the skip factors for individual tasks are generated as random variables from the interval $[1, S_{\max}]$. Fig. 17 shows the obtained QoS with respect to the maximum skip factor $S_{\max}$. RLP and BWP algorithms in combination with evolved heuristics outperform the conventional algorithms. It is interesting to notice that for $S_{\max} = 1$, BWP+GP-QoS-MA and RLP+GP-QoS-MA yield the same QoS as SRTF. Since all jobs are blue when $S_{\max} = 1$, the heuristics integrated in BWP+GP-QoS-MA and RLP+GP-QoS-MA have learned the behavior of SRTF. Moreover, when $S_{\max} = 1$, RTO algorithm skips all of the jobs and therefore $Q(\mathcal{T}) = 0$. Fig. 18 shows that only EDF and SRTF violations of the skip factor.

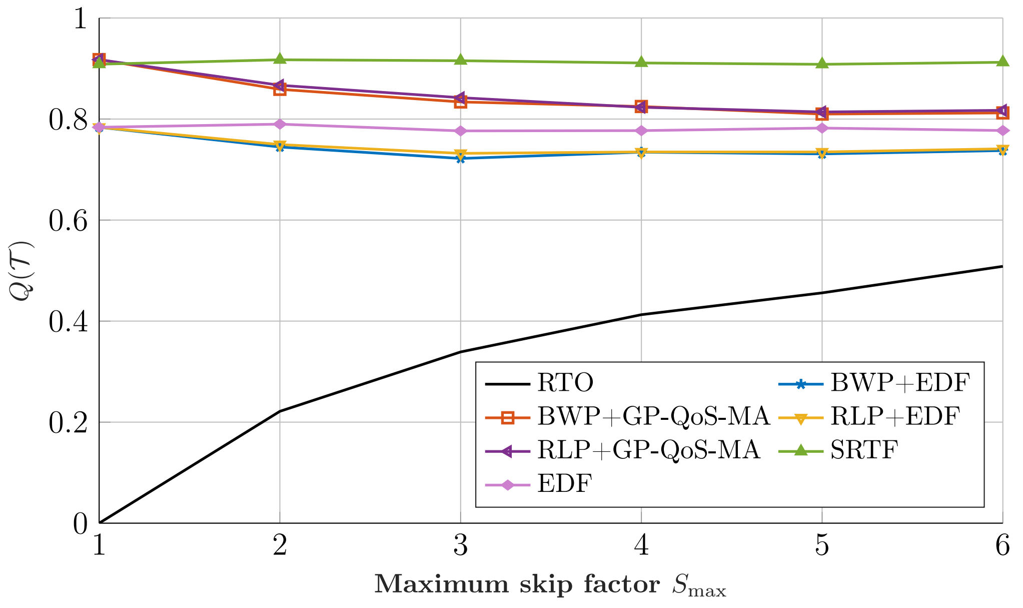


Figure 17: Average QoS with respect to the maximum skip factor $S_{\max}$.

In the second configuration, we assumed that all tasks in the set have the same skip factor. The skip factor was varied in the interval $[1, 4]$. For skip factors larger than 4, it was impossible to find feasible task sets with utilization factor 1.2. The result of the validation with respect to QoS are shown in Fig. 19. For $S_i = 1$, we can apply the same observations as in Fig. 17 for $S_{\max} = 1$, i.e., BWP+GP-QoS-MA and RLP+GP-QoS-MA yield the same QoS as SRTF and $Q(\mathcal{T}) = 0$ for RTO. We notice that BWP+GP-QoS-MA and RLP+GP-QoS-MA outperform conventional job-skipping algorithms for smaller skip factors with respect to quality of service. However, for skip factor $S_i = 4$, RLP and BWP algorithms combined with the evolved heuristics yield similar results as classical RLP and BWP. Note that even RTO achieved similar quality of service. This is due to the fact that with larger skip factors, there are fewer blue jobs released, and consequently there is less scope for improvement in terms of quality of service. Fig. 20 shows $N(\mathcal{T})$ with respect to skip factor S_i for all tasks. Once again, $N(\mathcal{T}) > 0$ for EDF and SRTF only.

An example of a QoS maximizing heuristic generated in our experiments is shown in (15).

$$\pi_{ij} = q_i \cdot (d_{ij} + (C_i - c_{ij})) \tag{15}$$

The term $(C_i - c_{ij})$ corresponds to the amount of computation that has been done by the current time instant for job $\mathcal{J}_{ij}$ and thus it decreases the priority of the jobs that have already been executing for a larger amount of time. Adding d_{ij} will decrease the priority of the jobs with the farthest deadline. The priority is also tuned with regard to the current quality of service of the job $\mathcal{J}_{ij}$. Higher priority is given to the job with lower quality of service.

Even though this heuristic is rather simple, it dominates conventional scheduling heuristics. However, there are more complex heuristics with even better performance, but they would be too challenging to interpret. It is important to emphasize that larger trees that correspond to complex heuristics often contain redundnant subtrees that can be removed. For instance, if a tree contains the subtree $d_{ij} + (C_i - C_i)$, it can be reduced to d_{ij}.

5.3.2. Evaluation of the GP-QoS-S approach

In the evaluation of the heuristics applied in GP-QoS-S approach, we analyze heuristics that are evolved specifically for a given task set, as described in Section 4. The performance of the heuristics is therefore validated on the same task sets that they were designed for, i.e., task sets used for training. Moreover, we compare the standalone heuristics to heuristics generated by GP-QoS-MA and integrated into job-skipping meta-algorithms. In these experiments, the GP-QoS-MA heuristics are evolved and validated in the same manner as in the GP-QoS-S approach: we consider a separate heuristic for each input task set and validate it on its corresponding task set.

In the validation process, parameters of the task sets, namely utilization factor u, maximum period value $T_{\max}$, number of task sets n, and maximum skip factor value $S_{\max}$ are selected as described in the previous experiments.

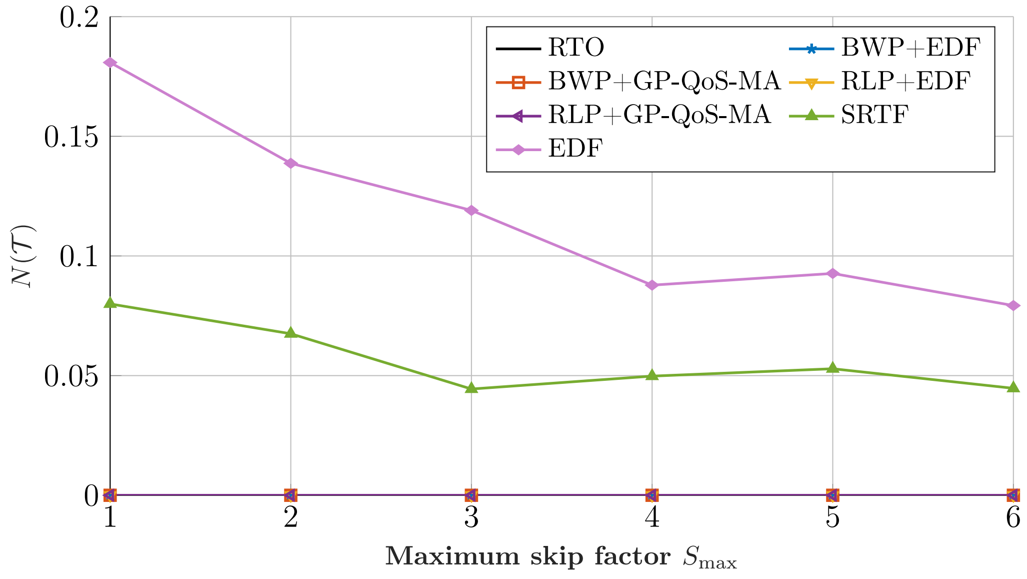


Figure 18: Average ratio of skip factor violations with respect to the maximum skip factor $S_{\max}$.

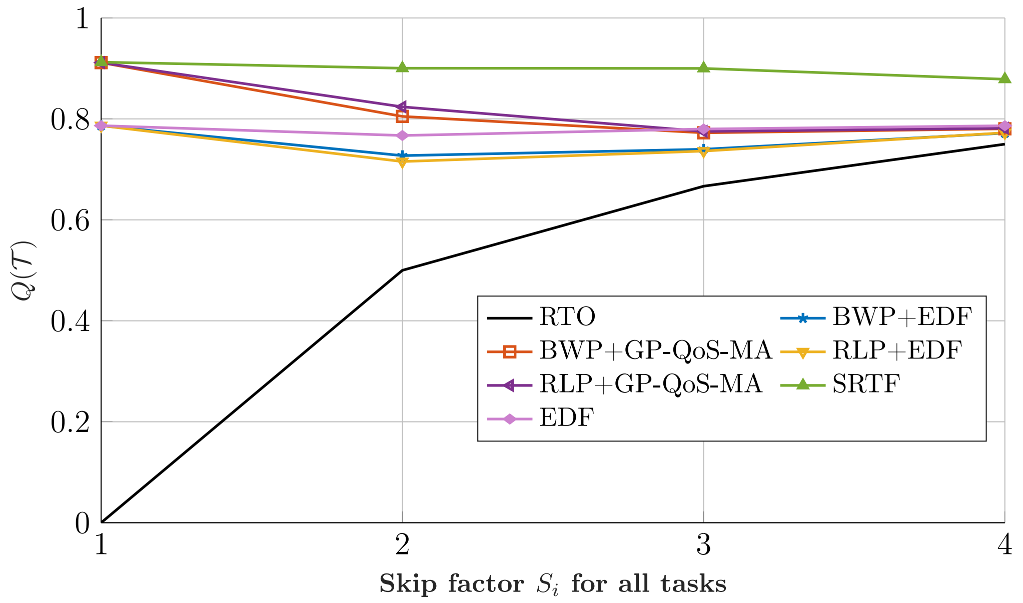


Figure 19: Average QoS with respect to the skip factor S_i for all tasks.

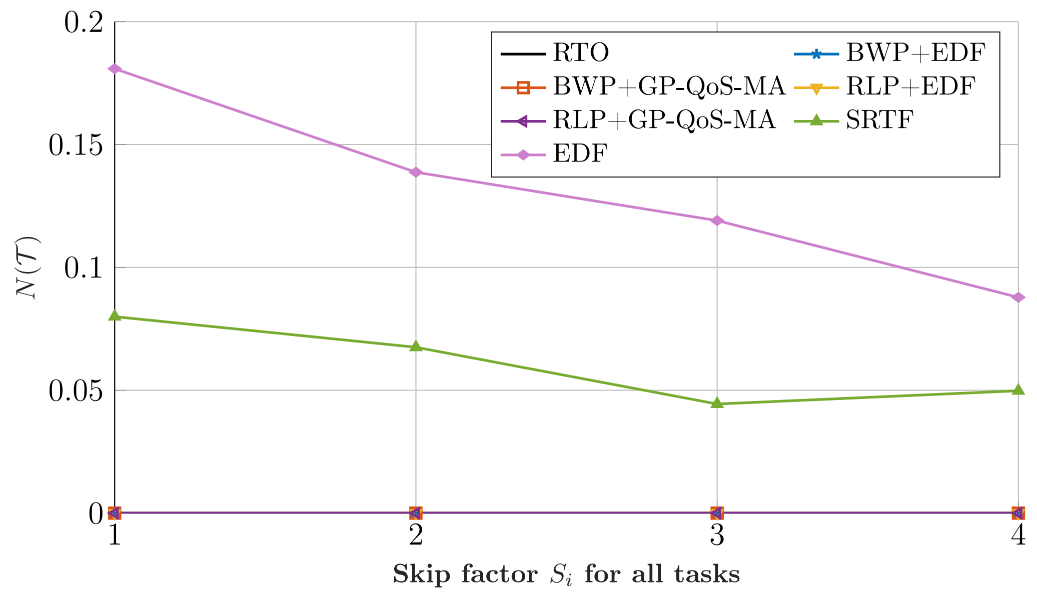


Figure 20: Average ratio of skip factor violations with respect to the skip factor S_i for all tasks.

For each considered parameter value, we generated 10 task sets and performed the evolution of the priority function for each task set. Fig. 21 shows the average quality of service obtained by the considered algorithms with respect to the utilization factor. We notice that generally, using the priority functions that are designed specifically for the task sets of interest can improve the QoS in comparison with the scenario where a single priority function is applied to task sets that are different from the sets used in training process. For instance, for utilization factor of 1.4, the QoS is increased by 3%. Moreover, BWP and RLP algorithms achieved optimality in underload conditions, which was not the case in the previous experiments. It is also important to notice that the heuristic which is evolved independently from the job-skipping meta-algorithms, denoted as *standalone* heuristic, performs better than the job-skipping algorithms combined with the evolved heuristics. Although the obtained QoS is still lower than the QoS obtained by SRTF, the standalone heuristic has achieved optimality in the underload conditions, and in Fig. 22 it is shown that the heuristic has learned guaranteed behavior—it produced zero violations of the skip factor. Fig. 23 shows a comparison of the GP-QoS-S approach and the previously discussed algorithms with respect to

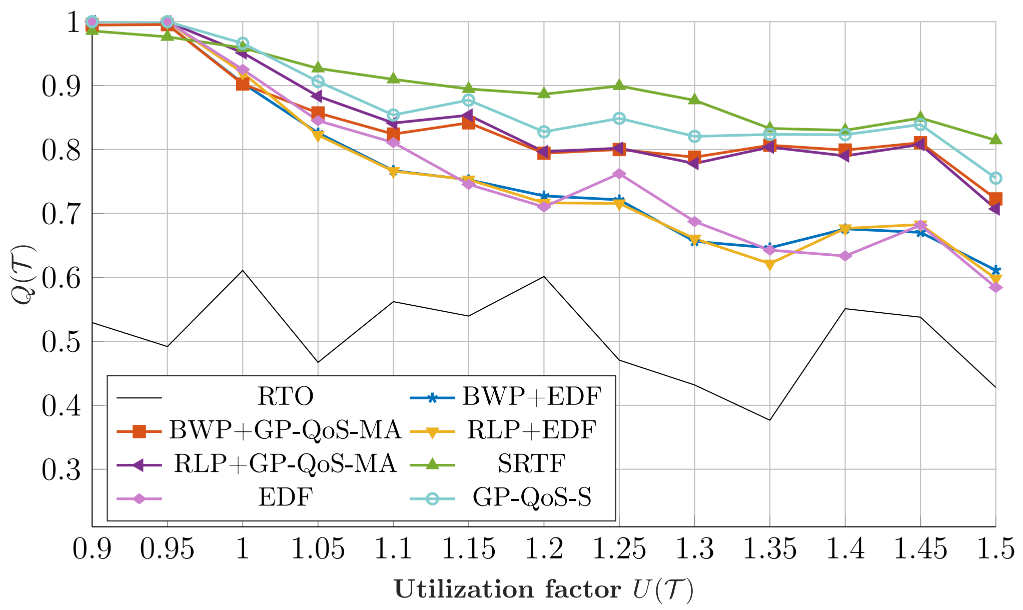


Figure 21: Average QoS with respect to the utilization factor $U(\mathcal{T})$, evaluated on the task sets used for training.

the maximum task period (T_{max}). We notice that GP-QoS-S consistently outperforms BWP+GP-QoS-MA and RLP+GP-QoS-MA by up to 10%. The results with respect to $N(\mathcal{T})$ metric are shown in Fig. 24. Once again, standalone heuristics produced no violations of the skip factor. In Fig. 25, the results of the observed algorithms in terms of QoS are compared with respect to the number of tasks in the set. It can be seen that GP-QoS-S method outperforms both conventional and GP-QoS-MA algorithms regardless of the task set size. Fig. 26 shows that heuristics evolved by GP-QoS-S showed guaranteed behavior on validation sets of different sizes. Fig. 27 shows the average QoS with respect to the maximum skip factor $S_{\max}$. It is interesting to notice that GP-QoS-S heuristics perform the same or better than SRTF for $S_{\max} = \{1, 2, 3\}$. In Fig. 28, we can see that job skipping algorithms, including GP-QoS-S heuristics, produced no skip factor violations.

The QoS for the configuration where all tasks have the same skip factor S_i is shown in Fig. 29, while Fig. 30 shows the results with respect to $N(\mathcal{T})$ metric. We notice an increase in the number of violations produced by the standalone heuristics. This is due to the fact that the scheduling problem is more complex if the weakly hard constraint of the tasks is more conservative

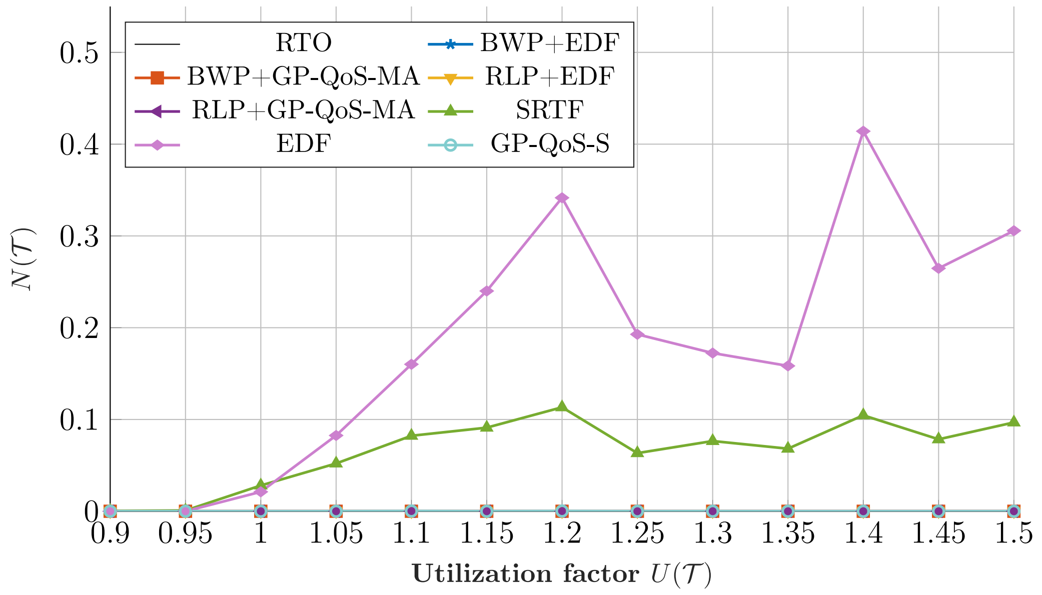


Figure 22: Average ratio of skip factor violations with respect to the utilization factor $U(\mathcal{T})$, evaluated on the task sets used for training.

and therefore the current configuration of the heuristics evolution process is not able to find a heuristic that satisfies the weakly hard constraint for all the task sets. This can be overcome by adjusting the parameters of genetic programming, e.g., increasing the tree depth or using a different stopping criterion for the evolution.

An example of a standalone priority function that maximizes the QoS, while guaranteeing the satisfaction of the weakly hard constraints for the given overloaded task set is shown in the following equation:

$$\pi_{ij} = \min(\sigma_{ij}, c_{ij} + \rho_{ij}) \tag{16}$$

The *min* operator with parameter σ_{ij} as operand ensures that the red jobs will have the highest priority—the value of the priority function π_{ij} will be 0. This expression shows that the heuristic has become aware of the state of the job (red or blue). For blue jobs, priority will be assigned as a sum of the remaining execution time c_{ij} and remaining time to deadline ρ_{ij}. Such heuristic for scheduling blue jobs is a combination of the SRTF and EDF heuristics. We can conclude that, given the proper inputs to the optimization process, the evolved heuristic can learn the behavior of the job-skipping meta-algorithm.

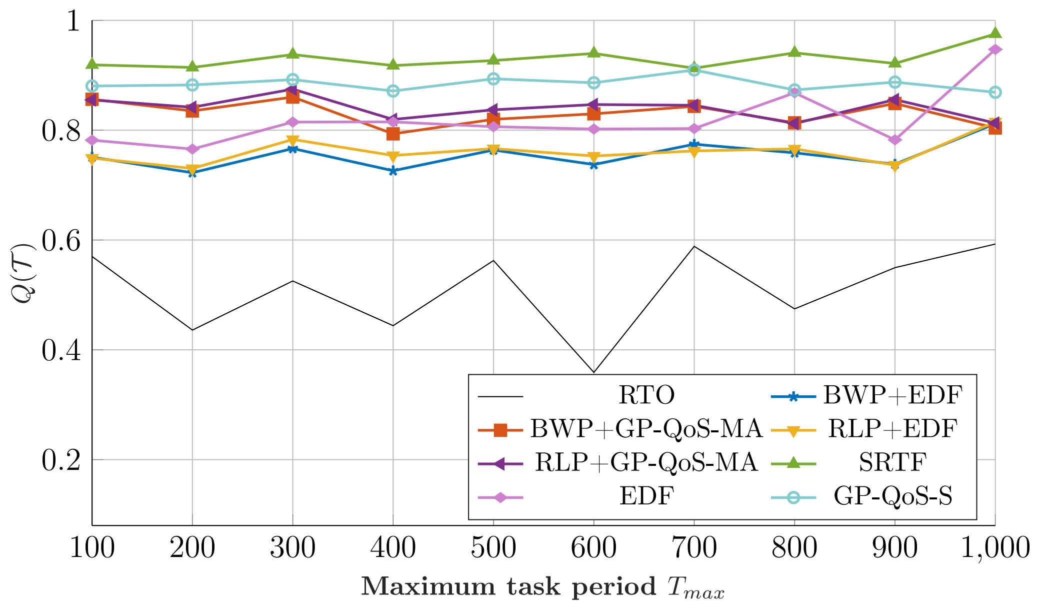


Figure 23: Average QoS with respect to the maximum task period $T_{\max}$, evaluated on the task sets used for training.

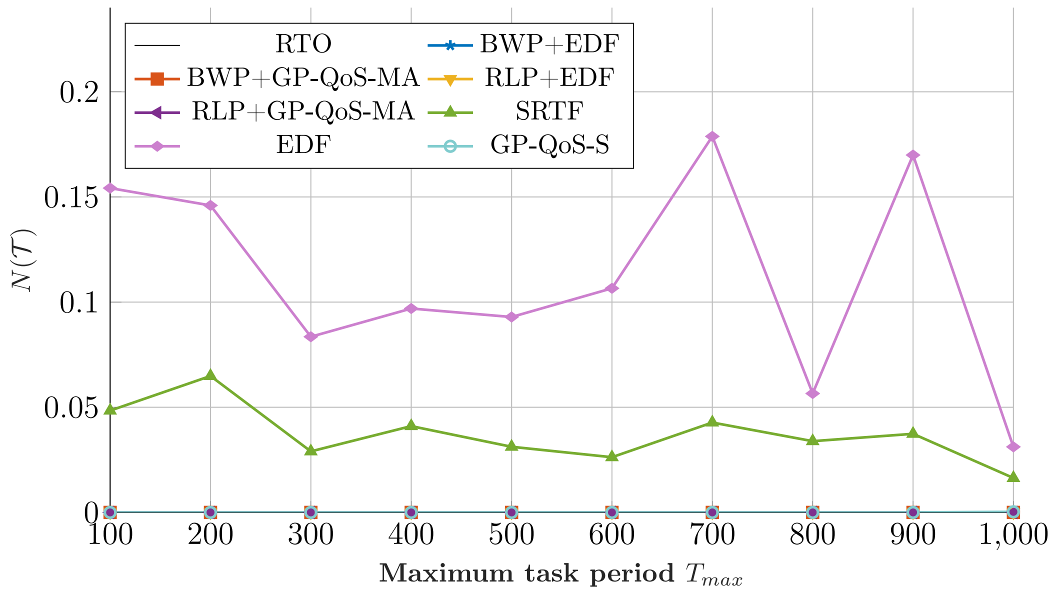


Figure 24: Average ratio of skip factor violations with respect to the maximum task period $T_{\max}$, evaluated on the task sets used for training.

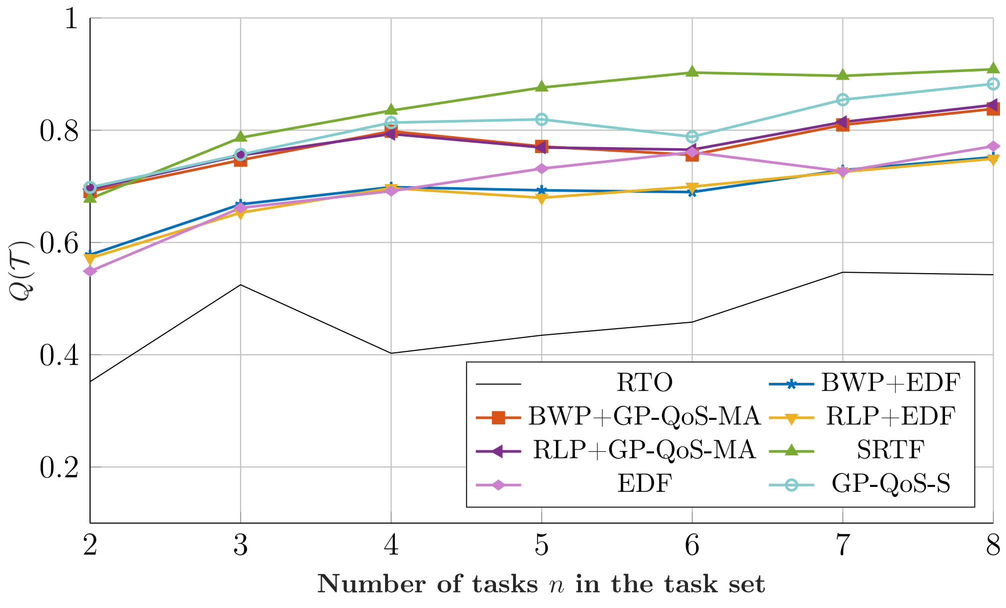


Figure 25: Average QoS with respect to the number of tasks N in the task set, evaluated on the task sets used for training.

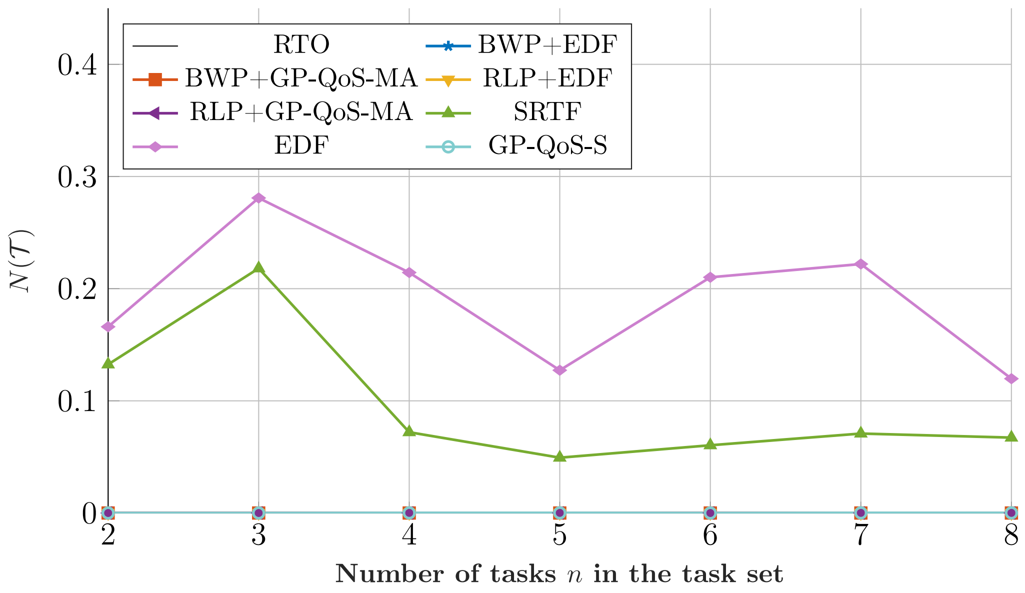


Figure 26: Average ratio of skip factor violations with respect to the number of tasks N in the task set, evaluated on the task sets used for training.

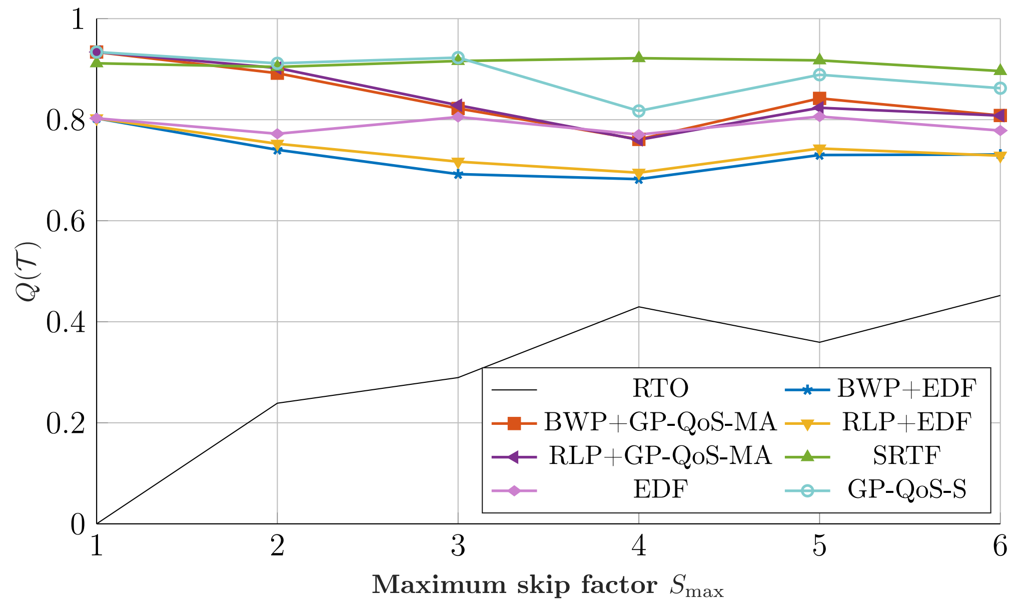


Figure 27: Average QoS with respect to the maximum skip factor $S_{\max}$, evaluated on the task sets used for training.

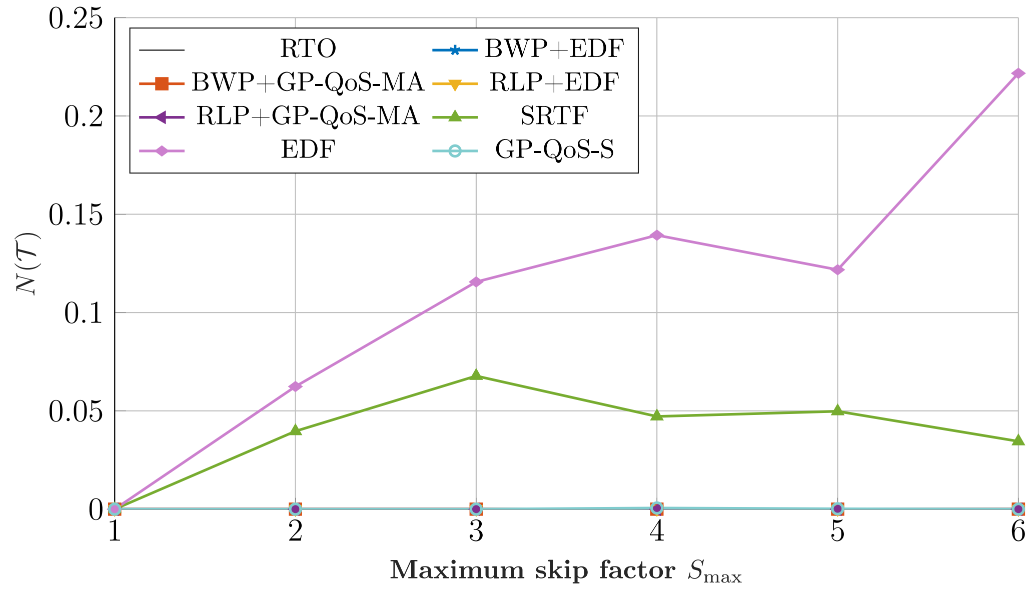


Figure 28: Average ratio of skip factor violations with respect to the maximum skip factor $S_{\max}$, evaluated on the task sets used for training.

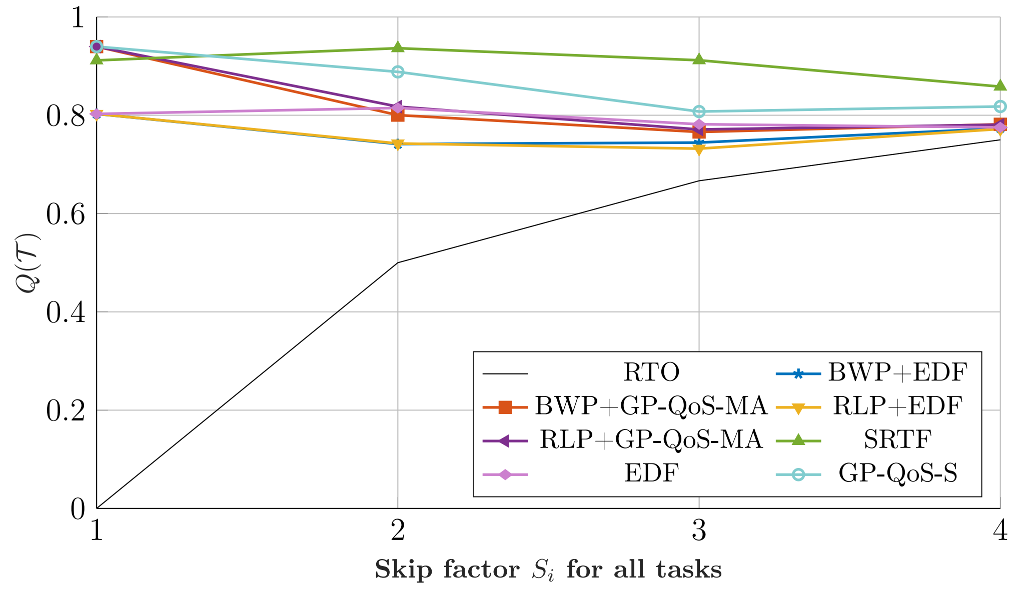


Figure 29: Average QoS with respect to the skip factor S_i for all tasks, evaluated on the task sets used for training.

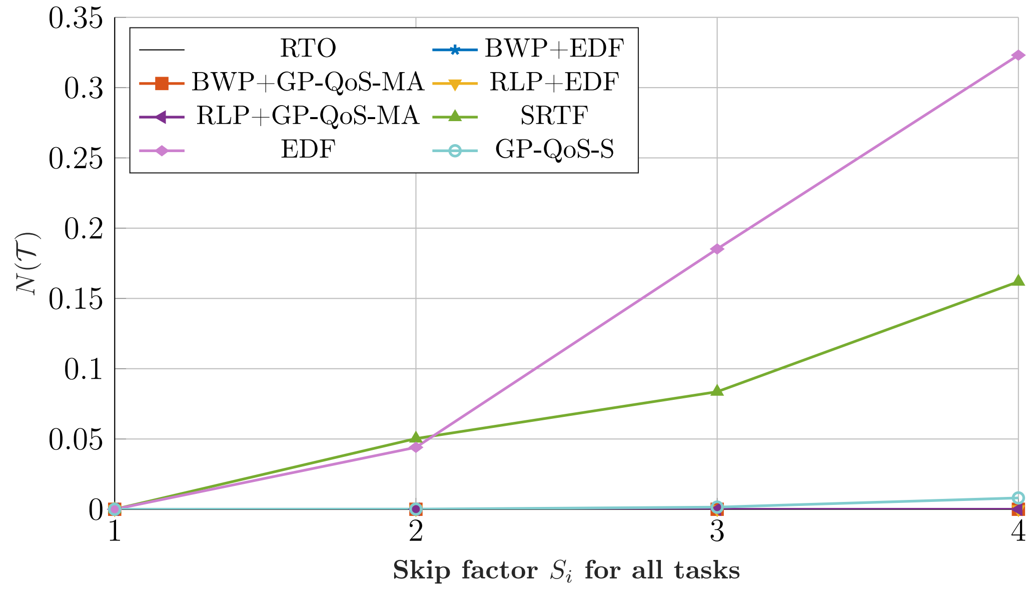


Figure 30: Average ratio of skip factor violations with respect to the skip factor S_i for all tasks, evaluated on the task sets used for training.

6. Applicability in embedded systems

In the presented approach, the skippable jobs are scheduled according to priorities assigned by priority functions. The process of evolving the priority functions is performed offline, and the resulting priority function can be embedded into the scheduler of a real-time operating system. During the execution of real-time task sets, the priorities of the skippable jobs are calculated by traversing a tree and the required number of operations is directly related to the depth of the tree. For instance, if a priority function corresponds to a tree with depth 3, the priority is calculated in $2^3 + 1 = 9$ operations. For trees of smaller depths, the time required for computing the priority is negligible with respect to the typical timing characteristics of real-time operating systems. For instance, the invocation of the scheduler in the FreeRTOS operating system by default occurs every millisecond.

In the work presented in [56], it is described how the FreeRTOS kernel can be modified in order to support dynamic priority assignment and the usage of custom scheduling heuristics. The scheduling heuristics generated by techniques desribed in this work can be easily incorporated into the FreeRTOS (or similar real-time operating systems) without introducing additional overhead. We suggest the implementation of dynamic scheduling with custom scheduling heuristics in RTOS kernel as follows. Upon every context switch, the scheduler calculates the priorities according to the given priority function for each job that is currently in the ready state. The job descriptors are inserted into a ready jobs list, sorted in increasing order of the priority value. As in the considered environment the lower priority value corresponds to the higher priority, the scheduler dispatches the job that corresponds to the descriptor at the head of ready jobs list. This mechanism is depicted in Fig. 31. The notations in the schematic are inspired by the terminology in FreeRTOS. The data structures that describe tasks are often referred to as *task control blocks*, which are denoted in the schematic as TCB. The TCB contains all the parameters of the tasks, as well as the dynamic parameters of the jobs, which are necessary for computing the priority. In the schematic, some of the parameters are omitted for simplicity. The priority function corresponds to the EDF algorithm: $\pi_{ij} = d_{ij}$.

Besides enabling the usage of optimized scheduling heuristics, another advantage of the presented approach is its modularity. If it is required to design heuristics for different scheduling environments, it is sufficient to modify the meta-algorithm in the evaluator. On the other hand, if a different metric

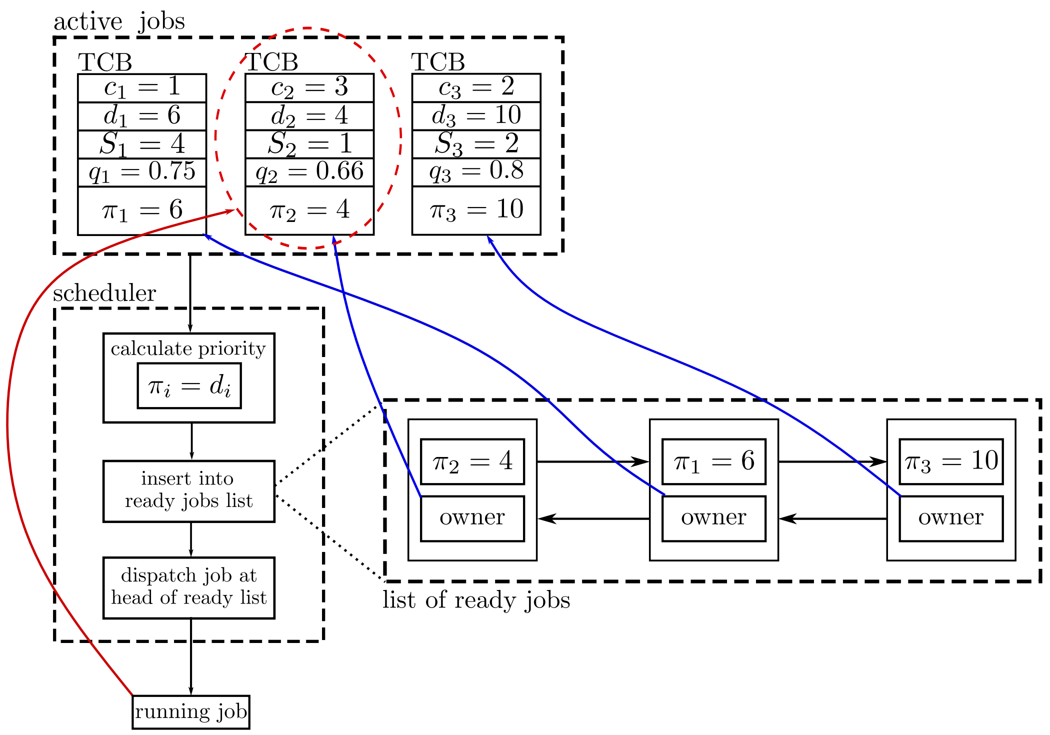


Figure 31: An example of a possible implementation of the considered scheduling environment in a real-time operating system.

must be optimized, it is only required to change the fitness function. For instance, if we need to optimize the scheduling heuristic in a soft real-time environment, we would use a different fitness function (e.g., average tardiness for all tasks in the set) and modify the evaluator in a way that the simulator corresponds to soft real-time instead of weakly hard real-time environment. Moreover, if we wanted to optimize multiple metrics, for instance, both quality of service and wasted processing time (CPU time wasted on executing skipped jobs), multi-objective optimization can be achieved with minimal changes in the evaluator.

7. Conclusion

In this paper, we presented a method for applying scheduling heuristics evolved by genetic programming for improving a custom performance metric in real-time systems with skippable jobs. The evolved heuristics are applied as functions for calculating the priorities of skippable jobs with the aim of increasing the quality of service in overload conditions. This scheduling environment corresponds to various applications, including real-time control systems, multimedia systems, and communication systems. We demonstrated a method for off-line design of efficient scheduling heuristics which can be customized in a straightforward way, depending on the target scheduling meta-algorithm. Moreover, we illustrated how the evolved heuristics can learn the behavior of the meta-algorithm if they are applied in a scheduling environment with predefined parameters of the task sets. The heuristics are suitable for on-line implementation in embedded systems which have limited resources due to their simplicity and computational efficiency.

In our experimental evaluation, we compared the performance of the evolved heuristics to the algorithms known from the literature (RTO, BWP, RLP). The heuristics that were encapsulated into BWP and RLP meta-algoritms and used for scheduling blue jobs achieved an improvement of the quality of service by up to 10% in comparison with the conventional algorithms. In a scenario where the heuristics are evolved specifically for considered task sets and are used as standalone scheduling algorithms, the QoS was improved by up to 15%. We conclude that this approach is suitable for optimizing custom performance metrics such as average quality of service in weakly hard real-time systems and it can also be applied for optimizing other metrics in different scheduling environments.

To conclude, our novel methods introduce several benefits. Firstly, the GP-QoS-MA method enhances the performance of the conventional job-skipping algorithms in terms of QoS. Secondly, the GP-QoS-S method provides both a mechanism for guaranteeing the fulfillment of weakly hard constraints and a best-effort scheduling rule that maximizes the QoS, integrated in a single heuristic. Thirdly, both methods can be easily customized for different scheduling environments and performance metrics. Finally, the methods are computationally efficient and therefore they are suitable for implementation in various real-time applications, including embedded systems.

A possible limitation of the presented approaches is that they maximize the quality of service of the entire task set, and do not regulate the deviations from the quality of service obtained per each task. To amend this, our future work will be focused on introducing multi-objective optimization in order to include additional metric, namely deviation from the defined quality of service for each task. The presented methods can also be further improved by integrating on-line acceptance tests for blue jobs in order to reduce processing time wasted on executing blue jobs that are eventually aborted. The acceptance tests could be evolved as separate heuristics using cooperative coevolution. This technique would perform with significantly lower overhead in comparison with conventional scheduling algorithms that employ on-line acceptance tests (RLP/T).

Acknowledgement

This work has been supported by the European Regional Development Fund under the project "A-UNIT - Research and development of an advanced unit for autonomous control of mobile vehicles in logistics" (KK.01.2.1.02.0119).

References

[1] G. C. Buttazzo, Hard real-time computing systems: predictable scheduling algorithms and applications, Vol. 24, Springer Science & Business Media, 2011.

[2] G. Bernat, A. Burns, A. Liamosi, Weakly hard real-time systems, IEEE transactions on Computers 50 (4) (2001) 308–321.

[3] G. Koren, D. Shasha, Skip-over: Algorithms and complexity for overloaded systems that allow skips, in: Proceedings 16th IEEE Real-Time Systems Symposium, IEEE, 1995, pp. 110–117.

[4] A. Marchand, M. Silly-Chetto, Rlp: Enhanced qos support for real-time applications, in: 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), IEEE, 2005, pp. 241–246.

[5] A. Queudet-Marchand, M. Chetto, Quality of service scheduling in the firm real-time systems, Real-Time Systems, Architecture, Scheduling, and Application (2012) 191.

[6] D. Jakobović, L. Budin, Dynamic scheduling with genetic programming, in: P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.), Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 73–84.

[7] D. Jakobović, K. Marasović, Evolving priority scheduling heuristics with genetic programming, Applied Soft Computing 12 (9) (2012) 2781 – 2789. doi:https://doi.org/10.1016/j.asoc.2012.03.065.

[8] M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 3rd Edition, Springer Publishing Company, Incorporated, 2008.

[9] S. Surono, K. W. Goh, C. W. Onn, A. Nurraihan, N. S. Siregar, A. Borumand Saeid, T. T. Wijaya, Optimization of markov weighted fuzzy time series forecasting using genetic algorithm (ga) and particle swarm optimization (pso), Emerg. Sci. J 6 (2022) 1375–1393.

[10] N. Kadhim, N. T. Ismael, N. M. Kadhim, Urban landscape fragmentation as an indicator of urban expansion using sentinel-2 imageries, Civil Engineering Journal 8 (09).

[11] J. Branke, S. Nguyen, C. W. Pickardt, M. Zhang, Automated design of production scheduling heuristics: A review, IEEE Transactions on Evolutionary Computation 20 (1) (2015) 110–124.

[12] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724.

[13] S. Nguyen, Y. Mei, M. Zhang, Genetic programming for production scheduling: a survey with a unified framework, Complex & Intelligent Systems 3 (1) (2017) 41–66. doi:10.1007/s40747-017-0036-x.

[14] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers & operations research 35 (10) (2008) 3202–3212.

[15] H. Nazif, L. Lee, A genetic algorithm on single machine scheduling problem to minimise total weighted completion time, European Journal of Scientific Research 35 (3) (2009) 444–452.

[16] I. Lee, J. Y. Leung, S. H. Son, Handbook of real-time and embedded systems, CRC Press, 2007.

[17] M. Hamdaoui, P. Ramanathan, A dynamic priority assignment technique for streams with (m, k)-firm deadlines, IEEE transactions on Computers 44 (12) (1995) 1443–1451.

[18] G. Buttazzo, S. Babamir, Handling overload conditions in real-time systems, in: Real-Time Systems, Architecture, Scheduling, and Application, Vol. 7, InTech, 2012.

[19] G. C. Buttazzo, G. Lipari, L. Abeni, Elastic task model for adaptive rate control, in: Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279), IEEE, 1998, pp. 286–295.

[20] W. Shih, J. Liu, J. Chung, D. W. Gillies, Scheduling tasks with ready times and deadlines to minimize average error, ACM SIGOPS Operating Systems Review 23 (3) (1989) 14–28.

[21] A. Shahzad, N. Mebarki, Learning dispatching rules for scheduling: A synergistic view comprising decision trees, tabu search and simulation, Computers 5 (1) (2016) 3.

[22] H. Ingimundardottir, T. P. Runarsson, Supervised learning linear priority dispatch rules for job-shop scheduling, in: International conference on learning and intelligent optimization, Springer, 2011, pp. 263–277.

[23] Y.-R. Shiue, Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach, International Journal of Production Research 47 (13) (2009) 3669–3690.

[24] G. R. Weckman, C. V. Ganduri, D. A. Koonce, A neural network job-shop scheduler, Journal of Intelligent Manufacturing 19 (2) (2008) 191–201.

[25] J. Branke, T. Hildebrandt, B. Scholz-Reiter, Hyper-heuristic evolution of dispatching rules: A comparison of rule representations, Evolutionary computation 23 (2) (2015) 249–277.

[26] Z. Hua, F. Qi, G. Liu, S. Yang, Learning to schedule dag tasks, arXiv preprint arXiv:2103.03412.

[27] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Genetic programming for job shop scheduling, in: Evolutionary and Swarm Intelligence Algorithms, Springer, 2019, pp. 143–167.

[28] M. Đurasević, D. Jakobović, K. Knežević, Adaptive scheduling on unrelated machines with genetic programming, Applied Soft Computing 48 (2016) 419–430.

[29] D. Jakobović, L. Jelenković, L. Budin, Genetic programming heuristics for multiple machine scheduling, in: European Conference on Genetic Programming, Springer, 2007, pp. 321–330.

[30] J. Park, S. Nguyen, M. Zhang, M. Johnston, Evolving ensembles of dispatching rules using genetic programming for job shop scheduling, in: European Conference on Genetic Programming, Springer, 2015, pp. 92–104.

[31] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Zhang, An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling, Applied Soft Computing 63 (2018) 72–86.

[32] C. Dimopoulos, A. M. Zalzala, Investigating the use of genetic programming for a classic one-machine scheduling problem, Advances in engineering software 32 (6) (2001) 489–498.

[33] C. Dimopoulos, A. Zalzala, A genetic programming heuristic for the one-machine total tardiness problem, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Vol. 3, IEEE, 1999, pp. 2207–2214.

[34] C. D. Geiger, R. Uzsoy, H. Aytuğ, Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach, Journal of Scheduling 9 (1) (2006) 7–34.

[35] T. P. Adams, Creation of simple, deadline, and priority scheduling algorithms using genetic programming, Genetic Algorithms and Genetic Programming at Stanford 2002 (2002) 84.

[36] F. J. Gil-Gala, C. Mencía, M. R. Sierra, R. Varela, Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time, Applied Soft Computing 85 (2019) 105782.

[37] I. Pavić, Optimization of schedulability and quality of service in real-time mixed-criticality systems. (2021).

[38] L. Palopoli, L. Abeni, G. Buttazzo, F. Conticelli, M. Di Natale, Real-time control system analysis: An integrated approach, in: Proceedings 21st IEEE Real-Time Systems Symposium, IEEE, 2000, pp. 131–140.

[39] C. C. Bissell, Control engineering, Chapman & Hall, London, 1994.

[40] C. Houpis, Digital control systems : theory, hardware, software, McGraw-Hill, New York, 1992.

[41] J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research 32 (11) (2013) 1238–1274.

[42] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971.

[43] R. A. Krohling, J. P. Rey, Design of optimal disturbance rejection pid controllers using genetic algorithms, IEEE Transactions on Evolutionary computation 5 (1) (2001) 78–82.

[44] Z.-L. Gaing, A particle swarm optimization approach for optimum design of pid controller in avr system, IEEE transactions on energy conversion 19 (2) (2004) 384–391.

[45] A. Pandey, R. K. Sonkar, K. K. Pandey, D. Parhi, Path planning navigation of mobile robot with obstacles avoidance using fuzzy logic controller, in: 2014 IEEE 8th international conference on intelligent systems and control (ISCO), IEEE, 2014, pp. 39–41.

[46] H. Asere, C. Lei, R. Jia, Cruise control design using fuzzy logic controller, in: 2015 IEEE International Conference on Systems, Man, and Cybernetics, IEEE, 2015, pp. 2210–2215.

[47] P. Ramanathan, Overload management in real-time control applications using (m, k)-firm guarantee, IEEE Transactions on parallel and distributed systems 10 (6) (1999) 549–559.

[48] A. Marchand, M. Silly-Chetto, Dynamic real-time scheduling of firm periodic tasks with hard and soft aperiodic tasks, Real-Time Systems 32 (1-2) (2006) 21–47.

[49] M. Caccamo, G. Buttazzo, Optimal scheduling for fault-tolerant and firm real-time systems, in: Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No. 98EX236), IEEE, 1998, pp. 223–231.

[50] A. Marchand, M. Chetto, Quality of service scheduling in real-time systems, International Journal of Computers, Communications and Control 3 (4) (2008) 354–366.

[51] H. Chetto, M. Chetto, Some results of the earliest deadline scheduling algorithm, IEEE Transactions on software engineering 15 (10) (1989) 1261–1269.

[52] M. Silly, The edl server for scheduling periodic and soft aperiodic tasks with resource constraints, Real-Time Systems 17 (1) (1999) 87–111.

[53] E. Bini, G. C. Buttazzo, Measuring the performance of schedulability tests, Real-Time Systems 30 (1-2) (2005) 129–154.

[54] Ecf - evolutionary computation framework, accessed: 2021-05-26 (2017).
URL `http://ecf.zemris.fer.hr/`

[55] M. O'Neill, Riccardo poli, william b. langdon, nicholas f. mcphee: a field guide to genetic programming (2009).

[56] K. Salamun, I. Pavić, H. Džapo, Dynamic priority assignment in freertos kernel for improving performance metrics, in: 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), IEEE, pp. 880–885.

Appendix A. Supplement to Example 3.1.

Table A.4 contains the priorities calculated by the priority function 8 of all active jobs upon every time instant that corresponds to idle time. The job that is dispatched

Table A.4: Priorities of blue jobs given by (8) for the schedule shown in Fig. 7.

t	π_i	t	π_i	t	π_i	t	π_i
0	$\pi_1 = 0.125$	16	$\pi_1 = 0.125$ $\pi_2 = 0.25$	24	$\pi_1 = 0.125$ $\pi_2 = 0.25$	34	$\pi_3 = 1$
8	$\pi_1 = 0.125$ $\pi_2 = 0.25$ $\pi_3 = 0.5$	18	$\pi_1 = 0.167$ $\pi_2 = 0.333$ $\pi_3 = 0.333$	25	$\pi_1 = 0.143$ $\pi_2 = 0.286$	36	$\pi_3 = 0.333$
9	$\pi_1 = 0.143$ $\pi_2 = 0.286$ $\pi_3 = 0.667$	19	$\pi_2 = 0.4$ $\pi_3 = 0.4$	26	$\pi_2 = 0.333$	40	$\pi_1 = 0.125$ $\pi_2 = 0.25$ $\pi_3 = 1$
10	$\pi_2 = 0.333$ $\pi_3 = 1$	20	$\pi_2 = 0.5$ $\pi_3 = 0.5$	30	$\pi_2 = 1$ $\pi_3 = 0.333$	41	$\pi_1 = 0.143$ $\pi_2 = 0.286$ $\pi_3 = 2$
11	$\pi_2 = 0.4$ $\pi_3 = 2$	21	$\pi_2 = 0.667$ $\pi_3 = 0.667$	31	$\pi_2 = 3$ $\pi_3 = 0.4$	42	$\pi_2 = 0.333$
12	$\pi_2 = 0.5$	22	$\pi_2 = 1$ $\pi_3 = 1$	32	$\pi_1 = 0.125$ $\pi_3 = 0.5$	43	$\pi_2 = 0.4$
13	$\pi_2 = 0.667$	23	$\pi_3 = 2$	33	$\pi_1 = 0.143$ $\pi_3 = 0.667$	44	$\pi_2 = 0.5$

is denoted in red. If there are multiple jobs with the same priority, the job with the earliest release time is dispatched. Although in practical applications it is unnecessary to compute priorities when there is only one active job, this case is covered in Table A.4 for completeness.

Table A.5 contains the priorities of all active jobs calculated by the priority function 9.

Table A.5: Priorities of jobs given by (9) for the schedule shown in Fig. 8.

t	π_i	t	π_i	t	π_i	t	π_i	t	π_i	t	π_i
0	$\pi_1 = 8$ $\pi_2 = 4$ $\pi_3 = 3$	8	$\pi_1 = 8$ $\pi_2 = 4$ $\pi_3 = 3$	16	$\pi_1 = 8$ $\pi_2 = 4$ $\pi_3 = 3$	24	$\pi_1 = 8$ $\pi_2 = 4$ $\pi_3 = 3$	32	$\pi_1 = 8$ $\pi_2 = 4$ $\pi_3 = 3$	40	$\pi_1 = 8$ $\pi_2 = 4$ $\pi_3 = 3$
1	$\pi_1 = 7$ $\pi_2 = 3.5$ $\pi_3 = 2.5$	9	$\pi_1 = 7$ $\pi_2 = 4$ $\pi_3 = 3$	17	$\pi_1 = 7$ $\pi_2 = 3.5$	25	$\pi_1 = 7$ $\pi_2 = 4$ $\pi_3 = 2.5$	33	$\pi_1 = 7$ $\pi_2 = 3.5$	41	$\pi_1 = 7$ $\pi_2 = 4$ $\pi_3 = 3$
2	$\pi_1 = 6$ $\pi_2 = 3$ $\pi_3 = 2$	10	$\pi_1 = 6$ $\pi_2 = 4$ $\pi_3 = 3$	18	$\pi_1 = 6$ $\pi_2 = 3$ $\pi_3 = 3$	26	$\pi_1 = 6$ $\pi_2 = 4$ $\pi_3 = 2$	34	$\pi_1 = 6$ $\pi_2 = 3$	42	$\pi_1 = 6$ $\pi_2 = 4$ $\pi_3 = 3$
3	$\pi_1 = 5$ $\pi_2 = 2.5$	11	$\pi_1 = 5$ $\pi_2 = 4$	19	$\pi_1 = 5$ $\pi_2 = 2.5$ $\pi_3 = 3$	27	$\pi_1 = 5$ $\pi_2 = 4$	35	$\pi_1 = 5$ $\pi_2 = 2.5$	43	$\pi_1 = 5$ $\pi_2 = 4$ $\pi_3 = 3$
4	$\pi_1 = 4$ $\pi_2 = 2$	12	$\pi_1 = 4$ $\pi_2 = 4$ $\pi_3 = 3$	20	$\pi_1 = 4$ $\pi_2 = 2$ $\pi_3 = 3$	28	$\pi_1 = 4$ $\pi_2 = 4$	36	$\pi_1 = 4$ $\pi_2 = 2$ $\pi_3 = 3$	44	$\pi_1 = 4$ $\pi_2 = 4$ $\pi_3 = 3$
5	$\pi_1 = 3$ $\pi_2 = 1.5$	13	$\pi_1 = 3$ $\pi_2 = 4$ $\pi_3 = 3$	21	$\pi_1 = 3$ $\pi_3 = 3$	29	$\pi_1 = 3$ $\pi_2 = 4$	37	$\pi_1 = 3$ $\pi_3 = 3$	45	$\pi_1 = 3$ $\pi_2 = 4$
6	$\pi_1 = 2$ $\pi_2 = 1$ $\pi_3 = 3$	14	$\pi_1 = 2$ $\pi_2 = 4$ $\pi_3 = 3$	22	$\pi_1 = 2$ $\pi_3 = 3$	30	$\pi_2 = 4$ $\pi_3 = 3$	38	$\pi_1 = 2$ $\pi_3 = 3$	46	$\pi_1 = 2$ $\pi_2 = 4$
7	$\pi_1 = 2$ $\pi_3 = 3$	15	$\pi_2 = 4$ $\pi_3 = 3$	23	$\pi_3 = 3$	31	$\pi_2 = 4$ $\pi_3 = 3$	39	$\pi_3 = 3$	47	$\pi_2 = 4$

Appendix B. Description of the simulator used for evaluation of the heuristics

The operation of the simulator is described by pseudocode in Alg. 4. The simulator registers three job events: release, completion, and skipping. Each event is tracked througout the simulation. Based on this information, the values of the given performance metric are computed at the end of the simulation. For instance, if quality of service is the metric of interest, it is necessary to register the number of released jobs and the number of completed jobs (line 10 and 16 in Alg. 4).

In each time instant, the highest-priority job is selected as the running job and its remaining execution time c_{ij} is decreased. The highest-priority job is determined according to the given scheduling meta-algorithm. In the case of GP-QoS-MA approach, the simulator executes a meta-algorithm that corresponds to either RLP or BWP, whereas for GP-QoS-S the meta-algorithm consists of priority calculation and dispatching only. The BWP and RLP meta-algorithms are implemented according to Alg. 1 and 2. At the end of the simulation, the value of the fitness function is computed based on the performance of the given individual in the simulation.

Algorithm 4 Simulation of task execution with job skipping.

```
Require: task set $\mathcal{T}$, scheduling meta-algorithm
Ensure: value of the performance metric at the end of the simulation
 1: initialize $\mathcal{T}$
 2: initialize queue of active jobs $\mathcal{Q}$
 3: for $t \leftarrow 0$ to hyperperiod do
 4:     for all jobs $\mathcal{J}_{ij}$ in $\mathcal{Q}$ do
 5:         if $c_{ij} = 0$ then
 6:             register $\mathcal{J}_{ij}$ as completed
 7:             remove $\mathcal{J}_{ij}$ from $\mathcal{Q}$
 8:     for all tasks $\tau_i$ in $\mathcal{T}$ do
 9:         if job $\mathcal{J}_{ij}$ is released then
10:             if job $\mathcal{J}_{i(j-1)}$ is still active then
11:                 register $\mathcal{J}_{i(j-1)}$ as skipped
12:                 remove $\mathcal{J}_{i(j-1)}$ from $\mathcal{Q}$
13:             initialize $\sigma_{ij}$, $c_{ij}$, $\rho_{ij}$
14:             register $\mathcal{J}_{ij}$ as released
15:             add $\mathcal{J}_{ij}$ to $\mathcal{Q}$
16:     if $\mathcal{Q}$ is not empty then
17:         get the highest-priority job according to the scheduling meta-algorithm
18:     for all jobs $\mathcal{J}_{ij}$ in $\mathcal{Q}$ do
19:         decrease $\rho_{ij}$
20:     decrease $c_{ij}$ for the highest-priority job
21: for all jobs $\mathcal{J}_{ij}$ in $\mathcal{Q}$ do
22:     register $\mathcal{J}_{ij}$ as skipped
23: calculate value of the performance metric
```

Appendix C. Analysis of the genetic programming parameters

In the parameter tuning process, we varied one parameter at a time while the other parameters remained fixed and set to their default values. We chose the default values for the parameters as follows:

- the number of generations was set to 30,
- the maximum genotype size (tree depth) was set to 8,
- the population size was set to 100.

We analyzed the impact of the observed parameters based on the results obtained by running 10 consecutive experiments and extracting the average performance of the best individuals in the population. The best individuals were evaluated on 1000 task sets for validation. The parameters of the task sets were set to default values as described in the previous subsection. In the parameter tuning process, we considered GP-QoS-MA approach, and a configuration where individuals are incorporated into the RLP meta-algorithm.

Firstly, we studied the impact of the number of generations as a stopping criterion for the evolution process. Fig. C.32 shows the average value of the fitness function, i.e., quality of service, obtained by the best individual of the population with respect to the number of generations. Note that the value of quality of service does not increase drastically

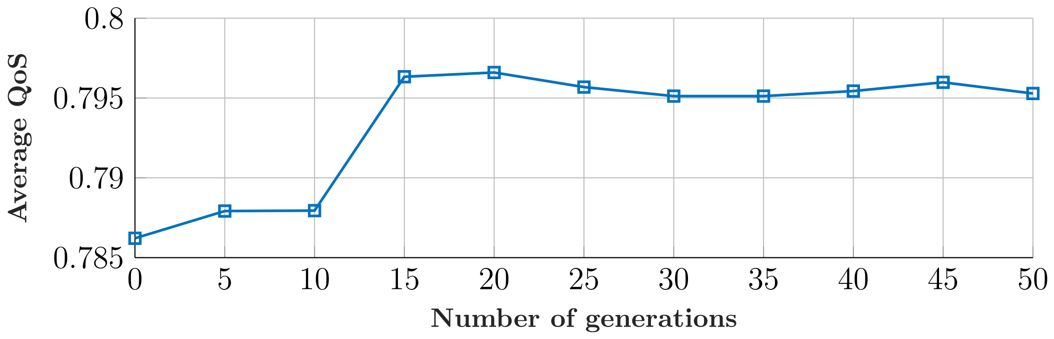


Figure C.32: Average QoS with respect to different number of generations. Generation 0 corresponds to the initial population.

after approximately 15 generations and therefore we chose 15 generations as the stopping criterion for the evolution. Secondly, we analyzed how varying the number of individuals in the population impacts the quality of service. In Fig. C.33, we notice that generally, the average quality of service is increased if the population size is increased. Clearly, increasing the population size causes an increase in the runtime of the training process and therefore we chose a population size of 100 as a trade-off. Finally, we examined the influence of the

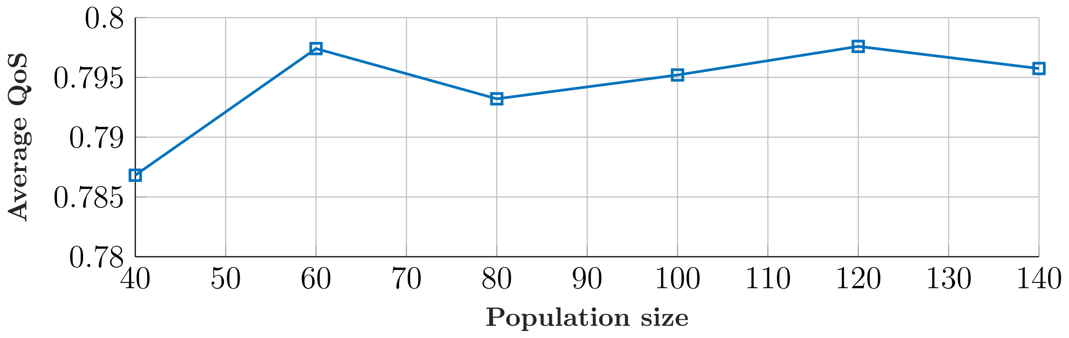


Figure C.33: Average QoS with respect to different population sizes.

maximum depth of the tree genotype. In this application, smaller trees are preferred over larger trees due to demands on the efficiency of priority function computation. Therefore, we considered trees with depths between 2 and 10. The infulence of tree depth on the average quality of service is depicted in Fig. C.34. We can note that increasing the tree

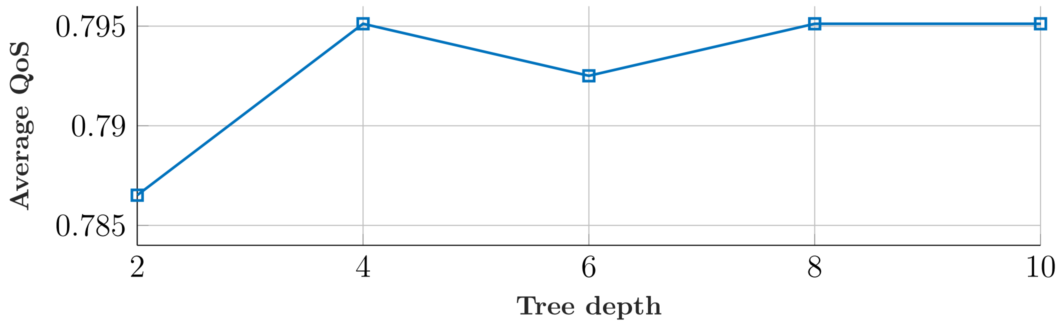


Figure C.34: Average QoS with respect to different tree depths.

depth from 2 to 10 increases the fitness value by less than 0.01. Since increasing the depth does not significantly improve the performance of the individuals, but it significantly increases the runtime of the evaluation, we chose a tree depth of 4 as the upper limit on the tree depth.

Appendix D. Statistical analysis of the proposed methods

In additional experiments, we performed statistical tests in order to verify that the results presented in Section 5 offer statistically significant improvement over the results achieved by standard scheduling algorithms.

Appendix D.1. Analysis of GP-QoS-MA approach

First, we analyzed the statistical differences in the performance of the RLP+GP-QoS-MA and BWP+GP-QoS-MA with respect to conventional RLP and BWP algorithms. Genetic programming was run for 30 times and we assessed the performance of the best individual from each run on the validation sets. Validation sets were generated with respect to parameters $U(\mathcal{T})$, N, T_{max}, S_i, and S_{max}, as described in Section 5. The best individuals from 30 runs performed similarly, which we confirmed by computing the correlation between the results of each individual. More precisely, we computed the Pearson's correlation coefficient of the QoS of the validation sets between each pair of the 30 best individuals. The minimal correlation coefficient that we obtained equals 0.98. Therefore, we concluded that it is sufficient to perform analysis on the individual from the first run.

Fig. D.35 shows the box plots obtained by evaluating the considered algorithms on the validation sets. Validation sets that are generated with respect to different parameters are shown on separate subplots. For brevity, GP-QoS-MA heuristics are denoted in the figure as *GP* and RLP+EDF and BWP+EDF are denoted as *RLP* and *BWP*, respectively.

The results shown in box plots coincide with the results from Section 5—algorithms with GP-QoS-MA dominate conventional algorithms in all of the performed experiments. It is important to notice that in experiments shown on Subfig. D.35a and D.35e, the outliers of the algorithms combined with GP-QoS-MA are still greater than the minimum (Q0) of the conventional RLP and BWP algoritms. Moreover, in the experiment shown in Subfig. D.35d, the outliers are greater than the lower quartile (Q1) of the conventional algorithms.

We also performed the Mann-Whitney test to compare the performance of the algorithm RLP+GP-QoS-MA with RLP+EDF and BWP+GP-QoS-MA with BWP+EDF. In the tests, we used the significance level $\alpha = 0.05$. If the Mann-Whitney test is run on all of the validation sets, the obtained p-value is approximately 0. Since in Section 5 the most extreme differences between the algorithms were obtained when evaluated with respect to different utilization factors, we ran the test separately on the sub-sets with the same $U(\mathcal{T})$. The obtained results are shown in Table D.6. For overloaded sets ($U(\mathcal{T}) > 1$) the obtained p-value equals 0. The p-values for sets with $U(\mathcal{T}) \leq 1$ indicate that there is no significant difference between RLP and BWP combined with evolved heuristics and conventional RLP and BWP, which is expected and it was discussed in Section 5.

Table D.6: P-values obtained by comparing RLP+GP-QoS-MA with RLP and BWP+GP-QoS-MA with BWP.

U($\mathcal{T}$)	**0.9**	**0.95**	1	**1.05**	**1.1**	**1.15**	**1.2**	**1.25**	**1.3**	**1.35**	**1.4**	**1.45**	**1.5**
RLP	0.84	0.99	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BWP	0.46	0.40	0.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

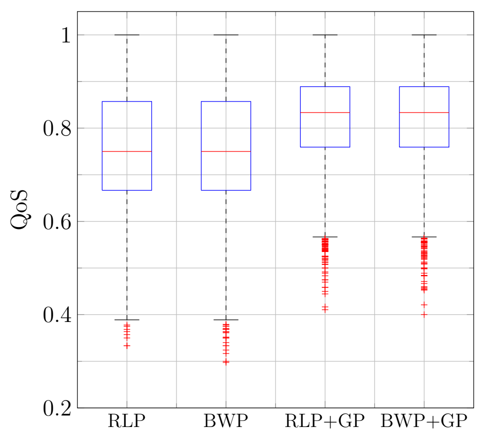
Appendix D.2. Analysis of GP-QoS-S approach

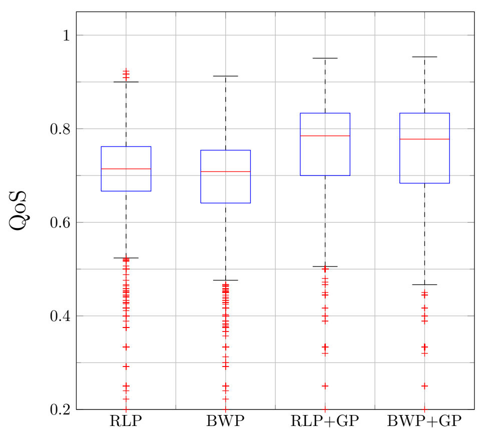
For testing the GP-QoS-S method, we generated validation sets in the same manner as described in Section 5. We performed the Mann-Whitney test with a significance level $\alpha = 0.05$. on separate subsets for each utilization factor. The results are summarized in Table D.7. The p-values for sets with $U(\mathcal{T}) > 1$ indicate that the obtained results are statistically significant, while the sets with $U(\mathcal{T}) \leq 1$ yield similar results as the GP-QoS-MA method.

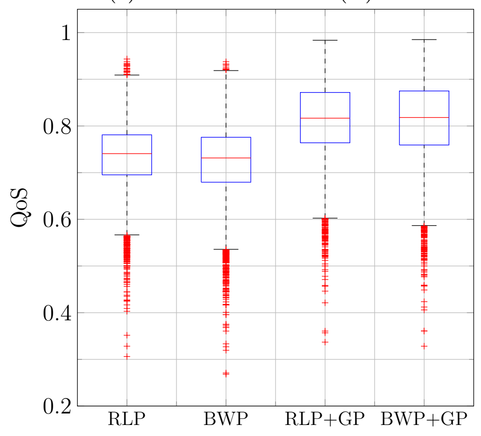
The differences between the GP-QoS-S heuristic and conventional algotihms is depicted as a scatter plot in Fig. D.36. QoS of sets scheduled with standalone heuristics are shown on y-axis, and the QoS of sets scheduled with conventional algorithms are shown on x-axis. The results with respect to each parameter for validation sets generation are shown on separate subplots. We notice that the data is grouped above the line QoS(standalone heuristics) = QoS(conventional algorithms), and thus we can conclude that the QoS obtained by GP-QoS-S approach is greater than or equal to the QoS obtained by conventional algorithms for all of the validation sets.

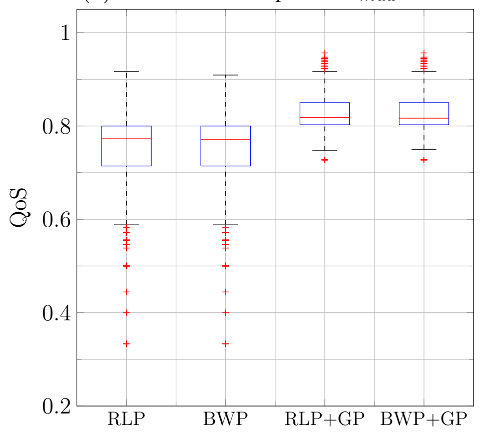
Table D.7: P-values obtained by comparing GP-QoS-S with RLP and BWP.

U($\mathcal{T}$)	**0.9**	**0.95**	1	**1.05**	**1.1**	**1.15**	**1.2**	**1.25**	**1.3**	**1.35**	**1.4**	**1.45**	**1.5**
RLP	0.42	0.29	0.13	0.04	0.00	0.04	0.03	0.00	0.00	0.00	0.00	0.00	0.00
BWP	0.42	0.29	0.13	0.02	0.00	0.03	0.02	0.00	0.00	0.00	0.00	0.00	0.00


(a) Utilization factor $U(\mathcal{T})$


(b) Maximum task period T_{max}


(c) Number of tasks in the set N


(d) Skip factor S_i for all tasks

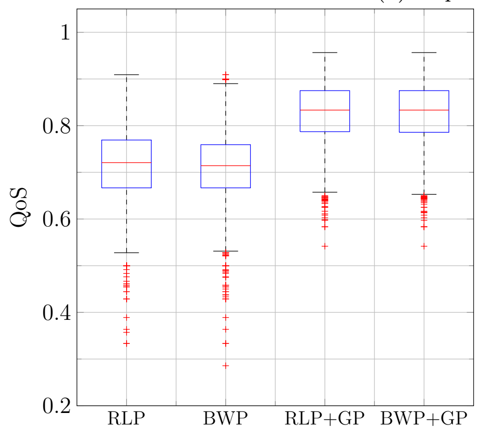
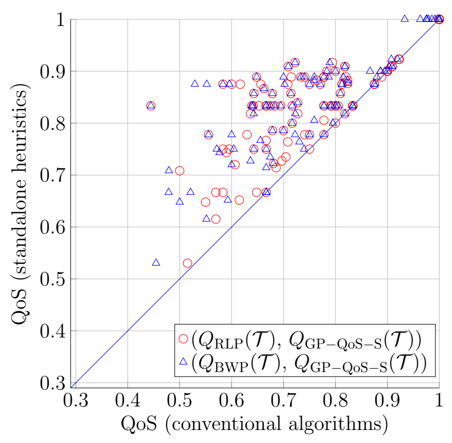
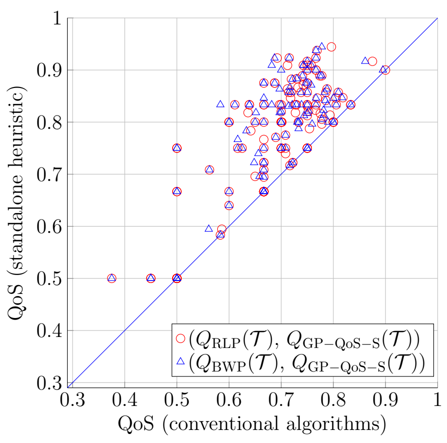

(e) Maximum skip factor S_{max}

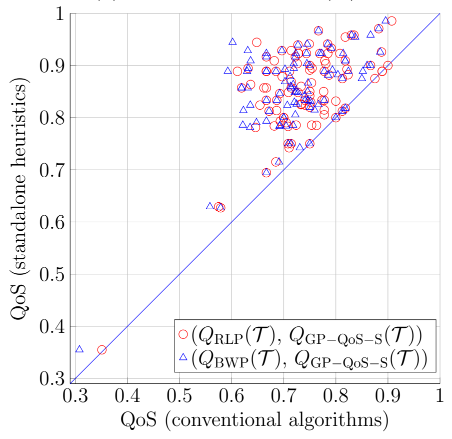
Figure D.35: QoS obtained by algorithms RLP+GP-QoS-MA, BWP+GP-QoS-MA, RLP+EDF, and BWP+EDF.



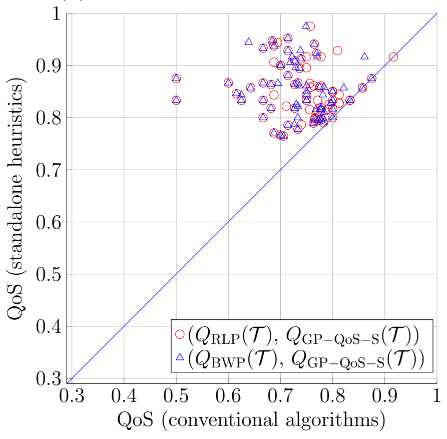
(a) Utilization factor $U(\mathcal{T})$



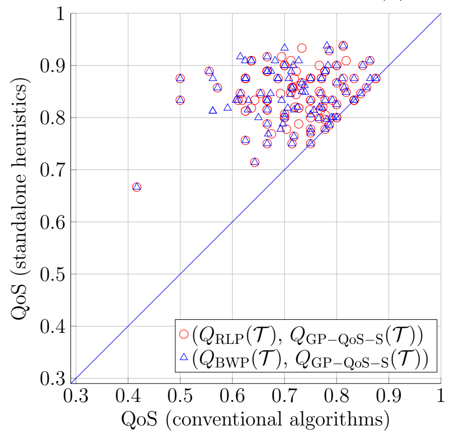
(b) Number of tasks in the set N



(c) Maximum task period T_{max}



(d) Skip factor for all tasks S_i



(e) Maximum skip factor S_{max}

Figure D.36: QoS obtained by GP-QoS-S, RLP, and BWP algorithms for task sets with different parameters.